Backbase Manual

Version 3.1.4

Thu Mar 30 11:56:27 CEST 2006

Table of Contents

Chapter 1 - Introduction
1.1 Backbase Rich Internet Applications 4
1.2 Introducing Backbase 4
1.3 Using the Examples in the Manual 13
1.4 Developer Resources 14
1.5 Runtime Developer Tools 18
Chapter 2 - Understanding BXML and BPC
2.1 BXML Overview 29
2.2 Backbase Presentation Client 34
2.3 Booting and Initialization 35
Chapter 3 - Introducing the BXML Tree
3.1 Managing the BXML Tree and DOM Tree 37
3.2 Targeting 38
3.3 Tree Manipulation 41
3.4 Loading 43
3.5 Submitting Forms 45
Chapter 4 - Asynchronous Execution
4.1 Asynchronous Execution 51
4.2 Locking For Explicit Synchronization 53
4.3 Aborting 54
Chapter 5 - Creating Application Layout
5.1 Defining Screen Partitioning 55
5.2 Backbase Controls Overview 57
5.3 Defining Layout Controls 57
5.4 Defining Navigation Controls 60
5.5 Defining Information Controls 68
5.6 Defining Chart Controls 71
5.7 Defining Form Controls 74
5.8 Defining Data Controls 78
5.9 Control Interaction 81
5.10 Adding Look And Feel (CSS) 83
5.11 Using Backbase Styled Controls 83
5.12 Overriding Backbase Styled Controls 85
5.13 Multimedia Elements 86
Chapter 6 - Applying User Interactions
6.1 Defining Events and Commands 87
6.2 Sequence of Event Triggering 91
6.3 Adding User Interaction Events 92
6.4 Defining Focus Model 94
6.5 Drag-And-Drop and Resizing 99
6.6 Adding Event Propagation 102
6.7 Adding Effects 105
6.8 Capturing Browser History and Bookmarking 106
Chapter 7 - Formalizing Patterns
7.1 Behaviors 109
7.2 User-Defined Controls 110
7.3 User-Defined Attributes 112
Chapter 8 - Defining Data Bindings
8.1 Templating 114
8.2 Browser Templating 117
8.3 Table Sorting 119

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 9 - Client-Side Data and Business Logic
9.1 Defining Task Execution 122
9.2 Client-Side Validation 122
9.3 Defining Variables 124
9.4 Using JavaScript 128
Chapter 10 - Appendix
10.1 Performance Tuning 131
10.2 XPath 2 131
10.3 Browser Compatibility 137

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 1 - Introduction

1.1 Backbase Rich Internet Applications

The objective of Backbase software is to enable web developers to create *Rich Internet Application* (RIA). In an RIA, the processing is done client-side, allowing you to build dynamic, interactive web sites that behave more like a desktop application than a traditional web application. To create an RIA, you therefore need a solution that deals with browser limitations and the limitations of HTML; interactive, dynamic, and customizable web applications require features that HTML and the browser platform do not support, such as being able to drag-and-drop elements.

Backbase provides the following solution to address these limitations:

- **Backbase Presentation Client** (BPC) - the BPC is a robust client-side AJAX engine that renders Rich Internet Applications at runtime. The BPC interprets the Backbase BXML tags and transforms them into the corresponding DOM structure
- **BXML Language** - BXML (**B**ackbase e**X**tensible **M**arkup **L**anguage) is Backbase's declarative, tag-based UI language, that extends XHTML. BXML tags and attributes are used to describe UI components, data structures, behaviors, events handling, and command execution. BXML is extensible; you can create your own controls, events, behaviors, and attributes.

Backbase uses AJAX, or **A**synchronous **Ja**vaScript and **X**ML, technologies to implement a Rich Internet Application. AJAX is a term describing a web development technique for creating interactive web applications using a combination of technologies; XHTML and CSS for presenting information, the Document Object Model manipulated through JavaScript to dynamically display and interact with the information presented, the XMLHttpRequest object to exchange data (using XML) asynchronously with the web server. AJAX applications send requests to the web server to retrieve only the data that is needed. The rendering of the data (the presentation), is generated client-side. The data to be rendered results in a partial update of the application, and not a full reload. The data-driven request-response model is implemented in a Backbase web application using the *Single Page Interface* (SPI) model.

1.2 Introducing Backbase

This objective of this section is to explain the essential concepts of the Backbase AJAX/RIA software to give you an overview of the technology before you start coding. The concepts are described in greater detail, with examples provided, in this manual.

AJAX software helps you develop more interactive web applications quicker. The toolkit is designed to overcome the following challenges:

- Browser incompatibilities
- Maintaining large pieces of JavaScript code
- Lack of standard (reusable) components
- Spending too much time on writing low-level 'plumbing' code
- Lack of development tools

The Backbase software offers a standard solution for all these challenges. Backbase does the following:

- Removes browser differences
- Remedies the shortcomings of HTML, CSS and JavaScript by using a robust execution system that has a simplified API (actually tags instead of an API)
- Provides managers and libraries to simplify common tasks
- Supports data bindings
- Creates an attractive user interface layer with feature-rich UI controls

1.2.1 Understanding BXML and BPC

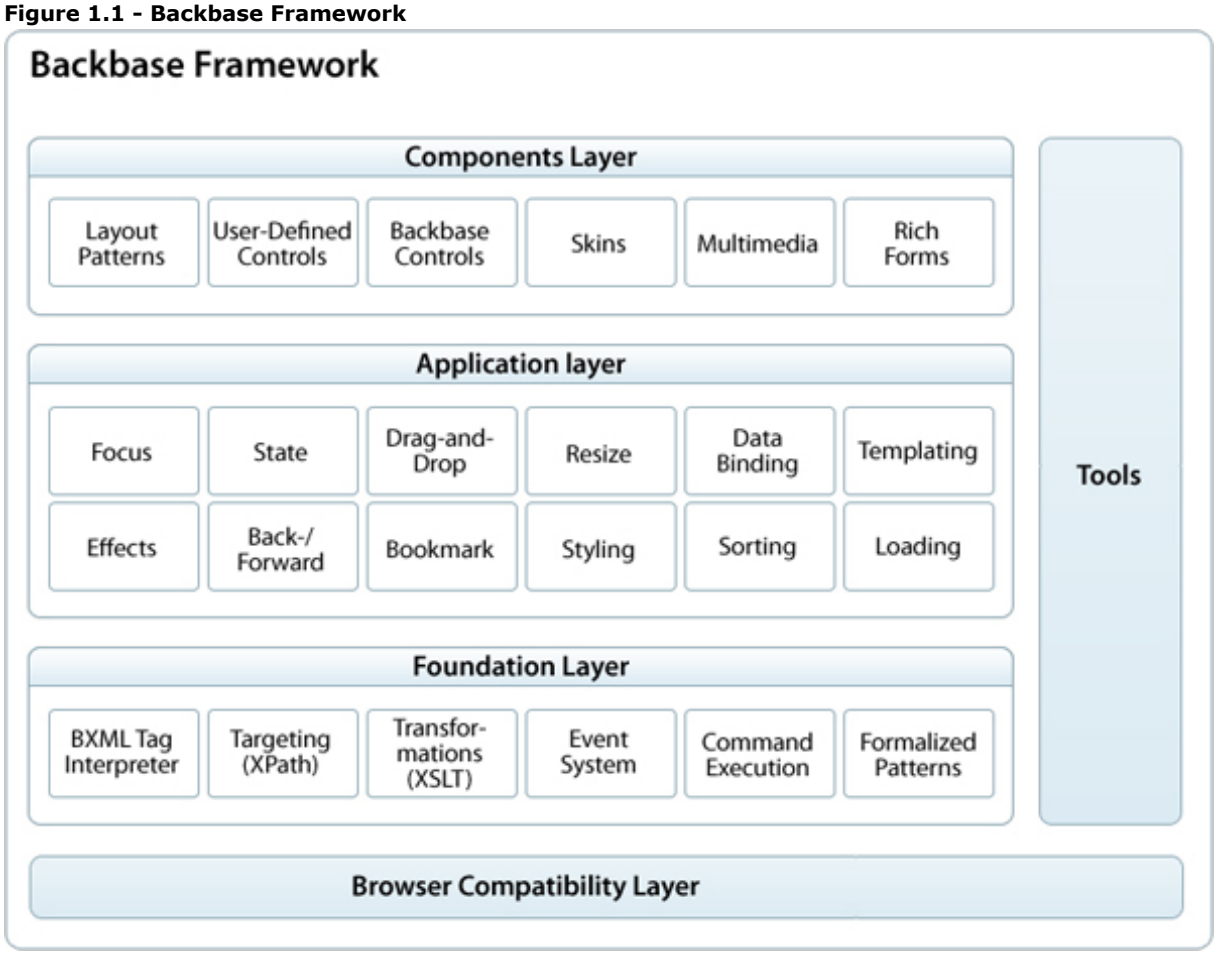
©2004-2006 Backbase B.V., All Rights Reserved.

The Backbase software contains two fundamental building blocks: the *Backbase Presentation Client* (BPC) and the *BXML* Language. BXML is Backbase's XML-based declarative language that extends HTML to provide dynamic client-side interaction and behavior. BXML tags and attributes are used to describe UI components, data structures, behaviors, event handling and command execution. BXML stands for Backbase eXtensible Markup Language, and it is a valid XML application. BXML uses two namespaces, indicated by `b:` and `s:`. The B-tags are usually visible to the end-users, for example `b:window` defines a control, and S-tags are system tags that are not visible, for example `s:task`. The BPC is the client-side runtime engine that interprets BXML tags and renders the result in the browser window. The BPC is a JavaScript application that is loaded automatically in the browser, completely transparent to the end-user. Once activated, the BPC translates the BXML at runtime into a standard DOM-tree (the DOM-tree is the browser's internal representation of a web page) that is rendered by the browser. Because BPC and BXML are based on native browser technology, existing DHTML technologies (XHTML, JS, CSS) remain fully functional within BXML pages. You can also freely mix BXML with XHTML and CSS.

1.2.2 Functional Overview

The following illustration provides an overview of the various layers in the Backbase software and their subcomponents:

Figure 1.1 - Backbase Framework



The layers are explained in the following sections, starting from the bottom up:

The **Browser Compatibility Layer** is intrinsic to the BPC. All you need to know about this layer is that it is present, and although you don't interact with it directly, it means you can safely develop without worrying about potential browser compatibility problems as that is all taken care of for you. For more information, see section **10.5 Browser**

©2004-2006 Backbase B.V., All Rights Reserved.

Compatibility.

The **Foundation Layer** is, as its name suggests, the foundation of the Backbase software that implements the declarative programming model. BXML tags are translated into native browser commands. As you are working on documents represented by tree structures (DOM and BXML Trees), you need a way to address parts of that tree: this is where XPath comes in. XPath is of fundamental importance in the XML world and is used almost everywhere in Backbase. It also provides an XSLT engine for transforming XML. This layer also includes a sophisticated Event System. Events trigger the execution of Commands. Events and Commands can be defined using Formalized Patterns, known as behaviors in Backbase, that provide better structuring and reuse of your code. Of course, you can also reuse your existing JavaScript code.

The **Components Layer** is the most visible part of user interfaces. It contains the Layout Manager, Backbase Controls, user-defined controls, and Rich Form Controls. The Layout provides the overall layout of the page (panels and windows). Backbase Controls are the interactive elements in an interface that combine styling and behavior, for example, a listview (sortable table), combobox, or slider. You can extend the Backbase Controls. Developers can also create their own controls or reuse controls made by others. These are known as User-Defined Controls. A special type of control are the Form Controls, which provide support for conditional form, input validation, and so on. You can also multimedia elements such as Flash, Java, and other media types that use plug-ins.

The **Application Layer** contains commonly used functionality for creating user interfaces. For example, the Focus Model is implemented in Backbase Controls to make the user interface navigable using the keyboard. For native web browser controls, the browser manages focus, but for DHTML controls, focus needs to be managed separately. This is the role of the Focus Engine. This layer also contains functionality for effects (animations), drag-and-drop, resizing, and form validation. Linking up with the browser backwards and forwards button and bookmarking is also supported, features are normally labor-intensive to implement. The color scheme and styling of borders, drop shadows, and so on, are part of the Styling sub-component. Loading XML data asynchronously is what is commonly referred to as 'AJAX'. These components take care of loading, manipulating, and binding of data. Backbase takes it several stages further by adding functionality for XSLT transformations, sorting, and binding data to Backbase Controls (user interface controls). Furthermore, the Component Layer can seamlessly access data from the Application Layer.

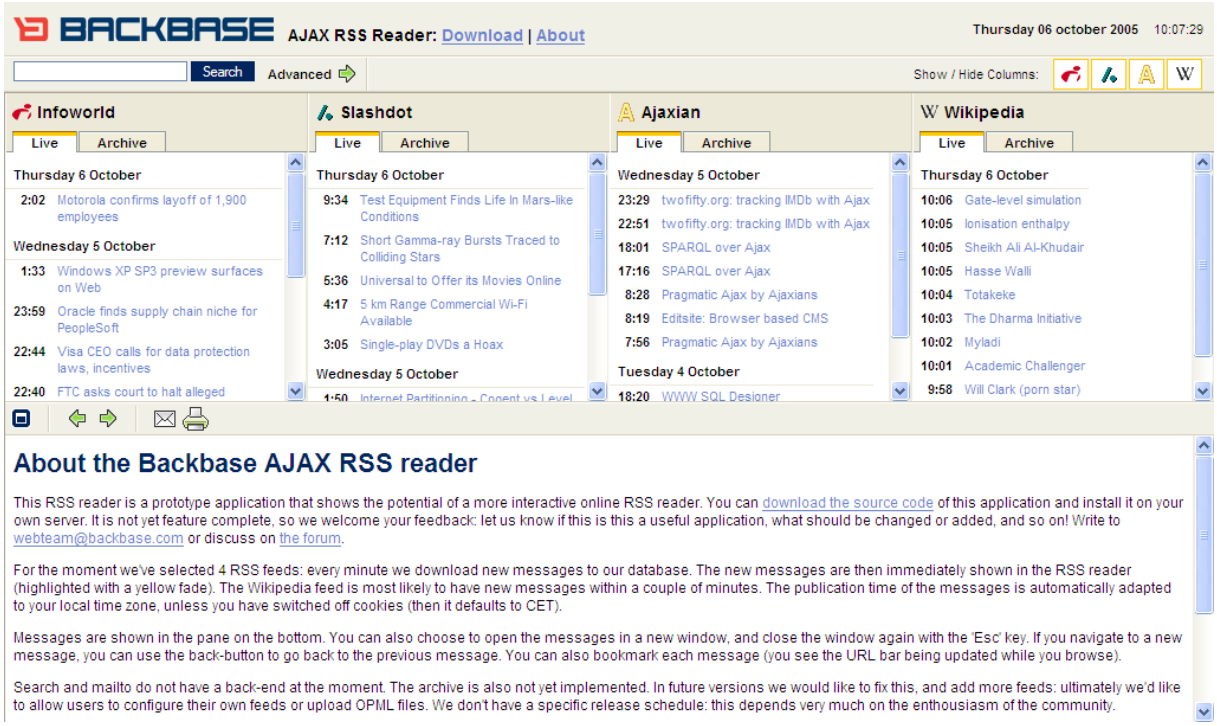
The **Tools Layer** consists of tools that developers need for effective development. Backbase has IDE plug-ins for Visual Studio, Eclipse, and Dreamweaver, as well as browser-based debugging and information tools.

1.2.3 Example: The RSS Reader

In this section, the Backbase RSS Reader is used to illustrate the functional overview. The RSS Reader is an application that displays 4 different RSS feeds and it shows new articles as they arrive:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.2 - RSS Reader


BACKBASE
AJAX RSS Reader: Download | About
Thursday 06 october 2005 10:07:29
Search
Advanced
Show / Hide Columns:
Infoworld
Live
Archive
Thursday 6 October
2:02 Motorola confirms layoff of 1,900 employees
Wednesday 5 October
1:33 Windows XP SP3 preview surfaces on Web
23:59 Oracle finds supply chain niche for PeopleSoft
22:44 Visa CEO calls for data protection laws, incentives
22:40 FTC asks court to halt alleged
Slashdot
Live
Archive
Thursday 6 October
9:34 Test Equipment Finds Life In Mars-like Conditions
7:12 Short Gamma-ray Bursts Traced to Colliding Stars
5:36 Universal to Offer its Movies Online
4:17 5 km Range Commercial Wi-Fi Available
3:05 Single-play DVDs a Hoax
Wednesday 5 October
Ajaxian
Live
Archive
Wednesday 5 October
23:29 twofifty.org: tracking IMDb with Ajax
22:51 twofifty.org: tracking IMDb with Ajax
18:01 SPARQL over Ajax
17:16 SPARQL over Ajax
8:28 Pragmatic Ajax by Ajaxians
8:19 Editsite: Browser based CMS
7:56 Pragmatic Ajax by Ajaxians
Tuesday 4 October
18:20 WWW SQL Designer
Wikipedia
Live
Archive
Thursday 6 October
10:06 Gate-level simulation
10:05 Ionisation enthalpy
10:05 Sheikh Ali Al-Khudair
10:05 Hasse Walli
10:04 Totakeke
10:03 The Dharma Initiative
10:02 Myladi
10:01 Academic Challenger
9:58 Will Clark (porn star)
About the Backbase AJAX RSS reader
This RSS reader is a prototype application that shows the potential of a more interactive online RSS reader. You can download the source code of this application and install it on your own server. It is not yet feature complete, so we welcome your feedback: let us know if this is this a useful application, what should be changed or added, and so on! Write to webteam@backbase.com or discuss on the forum.
For the moment we've selected 4 RSS feeds: every minute we download new messages to our database. The new messages are then immediately shown in the RSS reader (highlighted with a yellow fade). The Wikipedia feed is most likely to have new messages within a couple of minutes. The publication time of the messages is automatically adapted to your local time zone, unless you have switched off cookies (then it defaults to CET).
Messages are shown in the pane on the bottom. You can also choose to open the messages in a new window, and close the window again with the 'Esc' key. If you navigate to a new message, you can use the back-button to go back to the previous message. You can also bookmark each message (you see the URL bar being updated while you browse).
Search and mailto do not have a back-end at the moment. The archive is also not yet implemented. In future versions we would like to fix this, and add more feeds: ultimately we'd like to allow users to configure their own feeds or upload OPML files. We don't have a specific release schedule: this depends very much on the enthousiasm of the community.

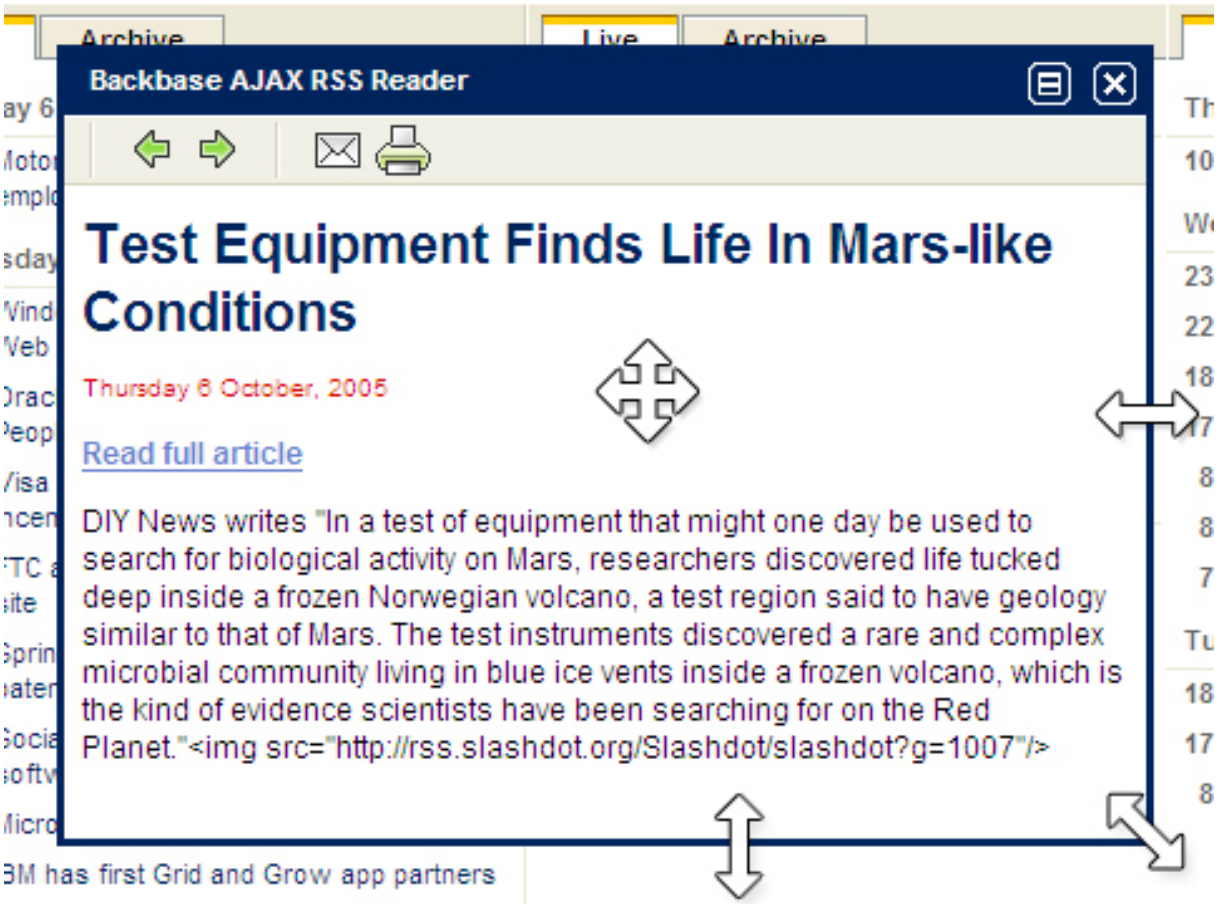
You can find this application online at `http://www.backbase.com/demos/RSS`.

Layout Layer - The layout of the RSS reader is a so-called *liquid* layout, meaning the screen resizes when you resize the browser window, and all columns resize by the same percentage. By using Backbase panels (`b:panelset` and `b:panel` tags), you can fix one or more columns at a certain width. This is also possible with CSS but is complex, and panels are simple to define (and they work similar to HTML framesets).

Another interesting feature of the RSS Reader is that you can change the message pane into a message Window by clicking on the top-left button. The Window is another concept that is natively supported by Backbase. You can also use operations from the Interaction Layer, such as dragging and resizing, but also closing, minimizing and restoring. The following screenshot shows the Window of the RSS Reader:

©2004-2006 Backbase B.V., All Rights Reserved.

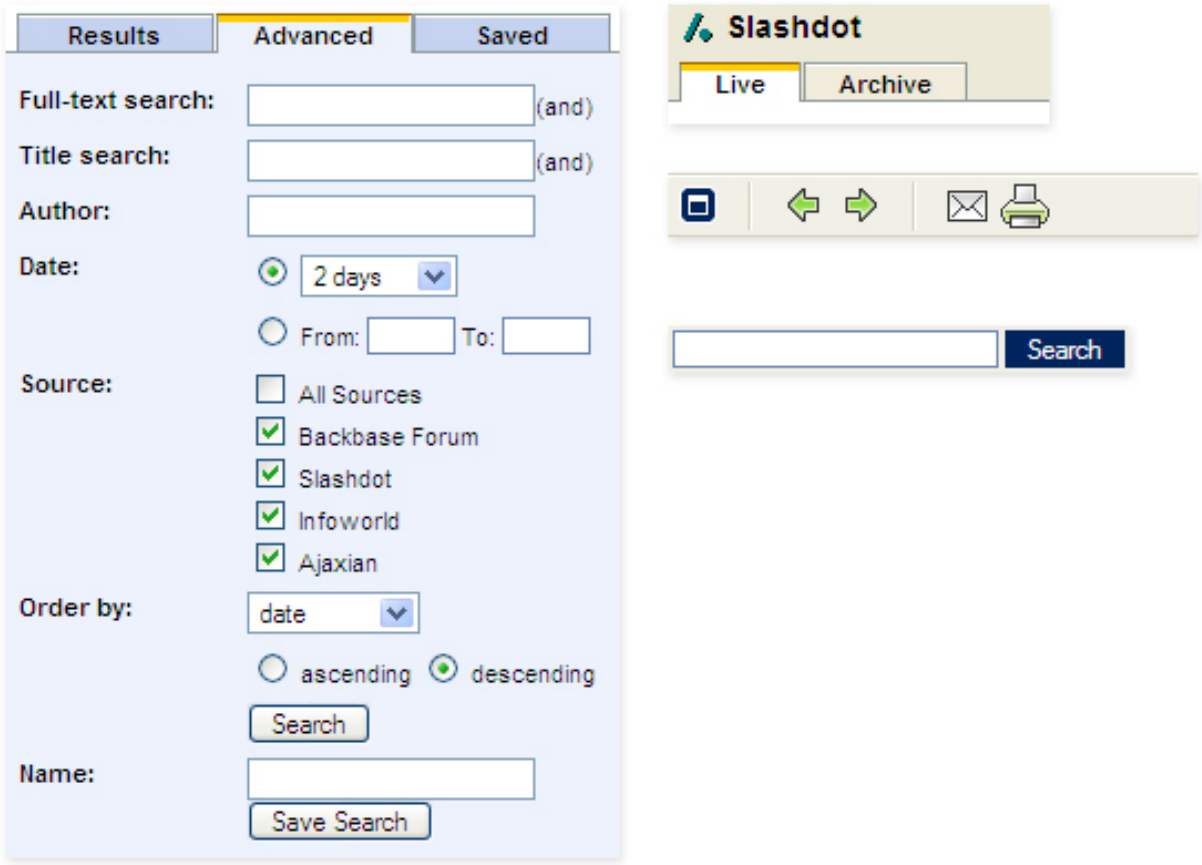
Figure 1.3 - Resizing


Backbase AJAX RSS Reader
Test Equipment Finds Life In Mars-like Conditions
Thursday 6 October, 2005
Read full article
DIY News writes "In a test of equipment that might one day be used to search for biological activity on Mars, researchers discovered life tucked deep inside a frozen Norwegian volcano, a test region said to have geology similar to that of Mars. The test instruments discovered a rare and complex microbial community living in blue ice vents inside a frozen volcano, which is the kind of evidence scientists have been searching for on the Red Planet."

Another important element of Layout are Controls. To get an idea of these controls, look at the examples in the Backbase Explorer. The Explorer is available on the web site (`www.backbase.com`), or in your Backbase installation directory. It features all the Backbase out-of-the-box controls, such as the following form elements, tabs, toolbar, and search box with button used in the RSS Reader example:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.4 - Control Features



User Interaction Layer - The User Interaction Layer focuses on the interaction in the application. This has a large impact on the overall usability. Navigation is a good example: if you browse through a couple of articles, you can simply use the browser back- and forward buttons to navigate between articles, implemented with a few lines of code. In addition, all articles can be bookmarked, in which case you can see that the URL has a special format:

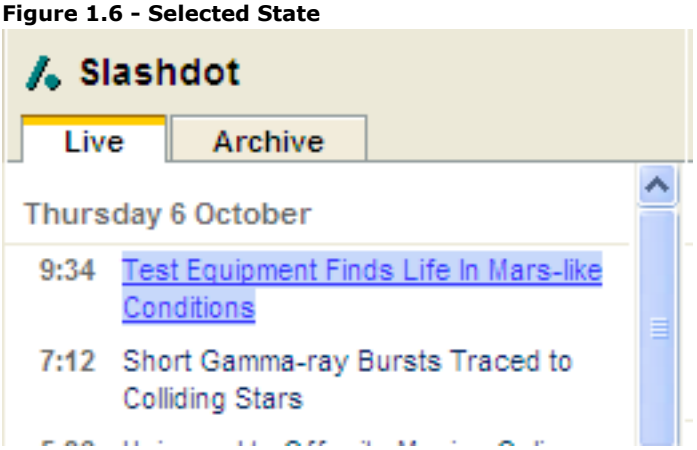
Figure 1.5 - Bookmarks

While you browse, the information behind the hash (#) is dynamically updated. The `aid` is the article ID, and `[10]` is the history index.

When you've selected an article, you see that it has a different color (see screenshot below). This is because the *State* of user interface elements is managed by Backbase. A selected item has the state *Selected* and this can be linked to different formatting (using behaviors).

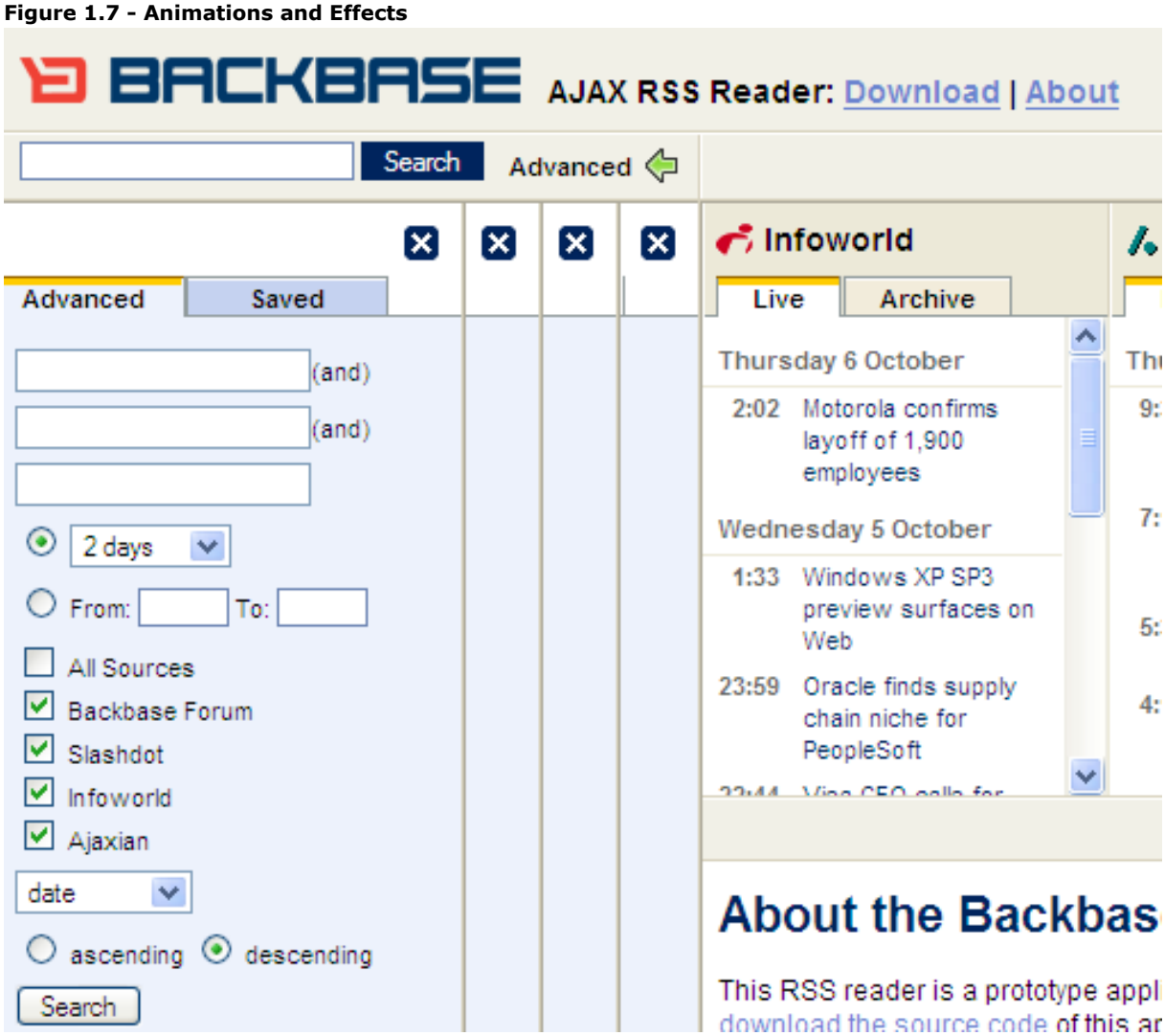
©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.6 - Selected State



Animations and Effects are also straightforward to implement. In the RSS Reader, several effects are used, for example a smooth opening of the search window. An animation cannot easily be captured in a screenshot, but the following illustration gives you an impression of what happens when you press the close button: the search pane slides away, while the article columns gradually expand.

Figure 1.7 - Animations and Effects



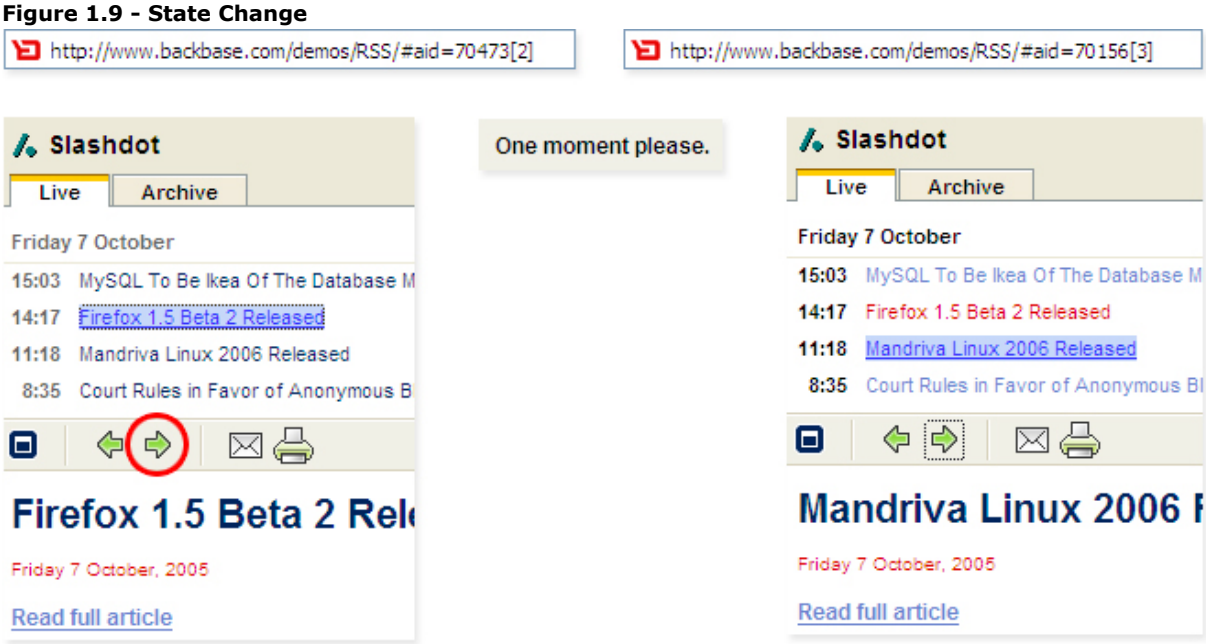
Obviously, functionality must be able to work together. Navigating between messages is a good example. On the toolbar, you find a 'newer article' and 'older article' buttons:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.8 - Newer and Older Article Naivigation Buttons

When you click on the 'older post' button, on the left you see the initial state, and on the right the state after the button has been pressed:

Figure 1.9 - State Change



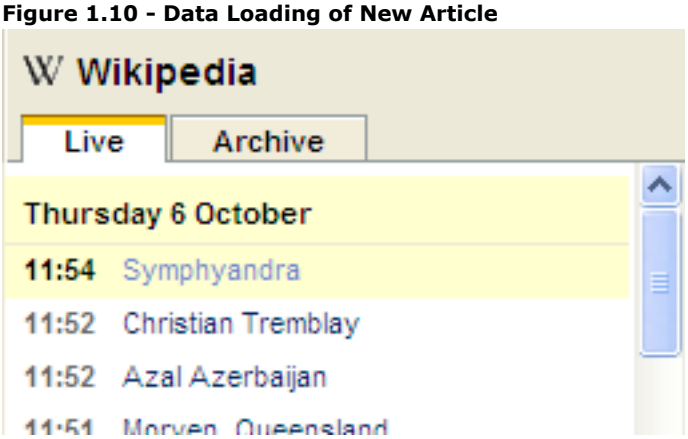
In the initial state, you see that the article is selected in the article list. This is because the state of the article is selected, which has a different style applied (a light blue background). When the button is pressed, the following happens:

- The new article is loaded in the article pane
- The state of the old article is set to deselected and becomes red
- The state of the new article is set to selected and it gets the blue background
- The URL is updated with a new Article identifier, and a new history index
- Therefore, the individual actions are triggering responses in different parts of the application, using the shared functionality from the Foundation and the Application Layers.

Data Layer - The RSS Reader also has some Data Loading actions. The most visible action is the loading of new articles. Every minute, the client application sends a request to the server to check whether there are new articles. If there are new articles, the summaries are sent back in a single XML file, the application copies the articles to the right-hand column, and a highlight effect is applied:

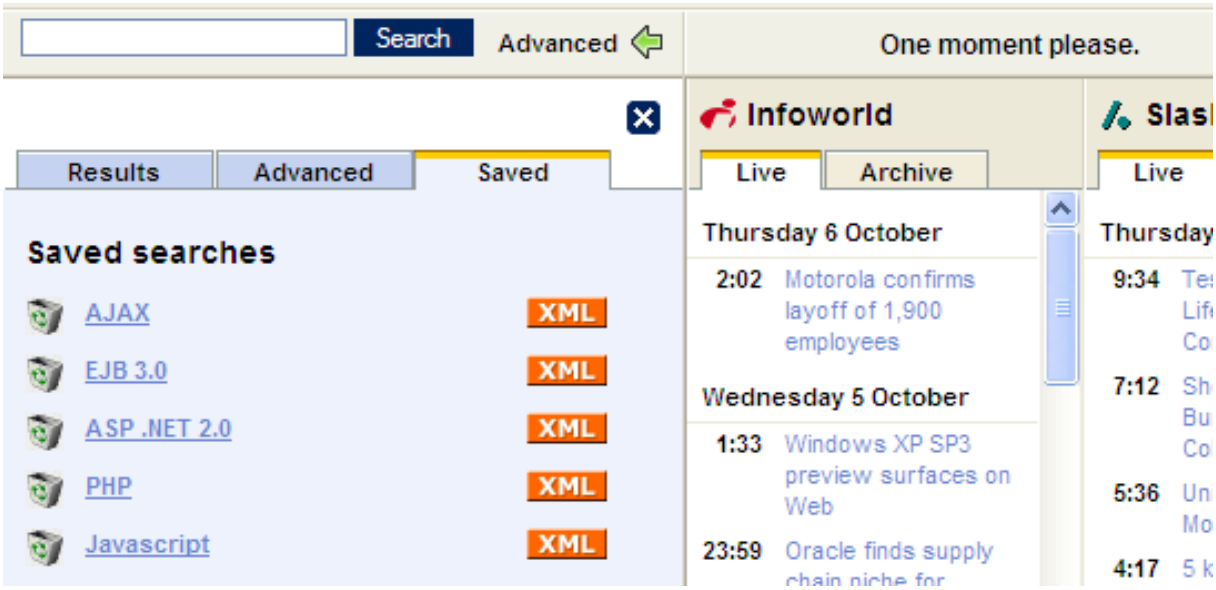
©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.10 - Data Loading of New Article



Backbase also provides functionality for loading on demand. For example, the advanced search panel is not loaded until it is selected. Once it is loaded, it is stored in the cache and is not reloaded. Of course, you can also specify that a screen should be reloaded every time it is selected, or that it is reloaded periodically. When the system is loading, it is possible to display a loading message, as the following screenshot demonstrates (see the message in the top right-hand corner):

Figure 1.11 - Loading Message

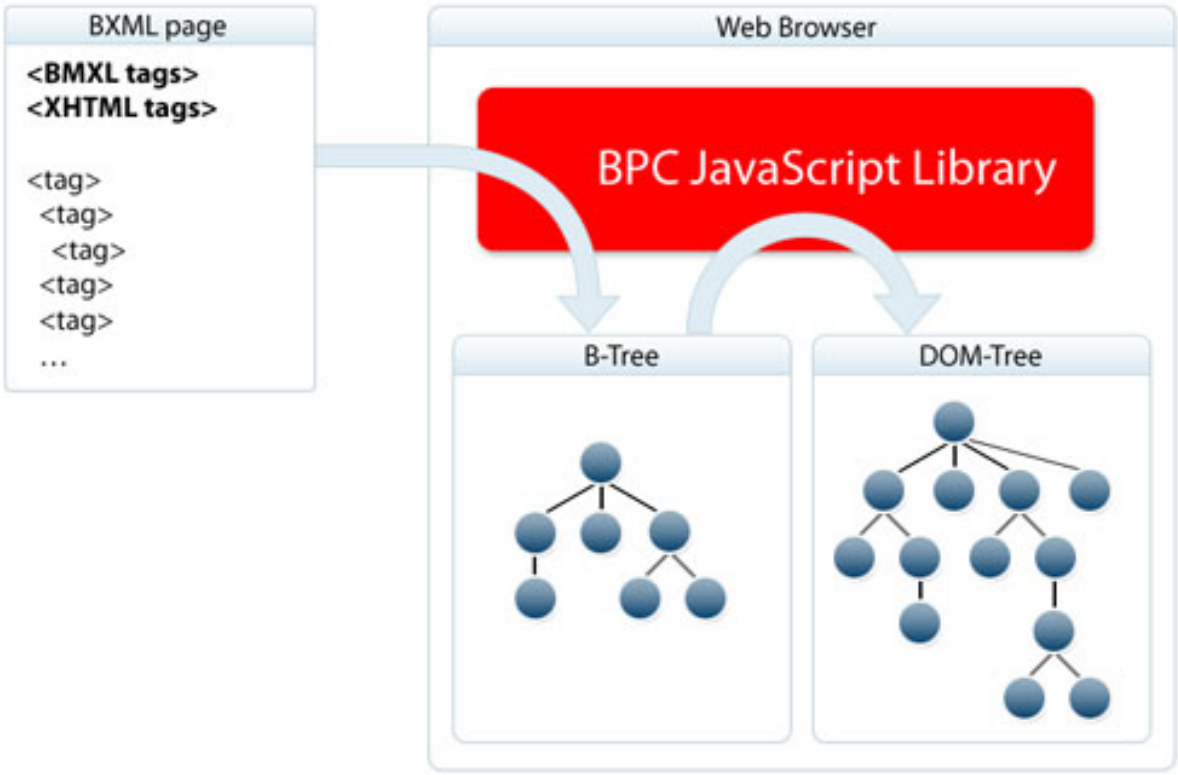


1.2.4 Introducing the BXML Tree

After reading the previous sections, you should by now have an overview of the functionality provided by Backbase software. This section delves deeper into the core of Backbase to give an overview of the rendering model and explain how BXML tags are loaded, interpreted, and executed. This corresponds with the *BXML Tag Interpreter* and *Targeting* components in the functional overview.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.12 - BXML Tree and DOM Tree



Web browsers do not recognize BXML tags, therefore the Backbase Presentation Client (BPC) translates BXML to native browser commands. A good way to visualize this is to think of two trees as shown in the illustration below: one tree is comprised of BXML, and the other of HTML (the DOM-tree). The BPC translates the BXML Tree into a standard DOM Tree, which is rendered by the browser.

A Backbase application modifies the BXML tree whenever something needs to change in the user interface. This is what's know as Tree Manipulation. You can load new BXML elements into the tree, remove them, copy them, transform them, and so on. To illustrate this, let's have a look at drag-and-drop: if a user drags an item, what is actually happening is that they are dragging part of the BXML tree. When the item is dropped in another location, this XML fragment is inserted in the BXML Tree in the new location.

One of the first questions that you may have is "how do I select or address part of the tree?" You do this by what is known as *targeting* using the XPath language. XPath is standardized by the W3C, and Backbase has implemented XPath version 2.0. For more information, see the sections **3.2 Targeting** and **3.3 Tree Manipulation**.

XPath is not only used for targeting parts of the BXML and DOM-trees, but it can also perform a variety of functions, such as string manipulation and calculations. Backbase also introduces BPC-specific extensions to XPath. For more information, see the Appendix or the BXML Reference PDF.

1.3 Using the Examples in the Manual

This manual describes the concepts of application development with Backbase, and provides examples that you can cut-and-paste into a start-up page to put the theory into practice and try out the code for yourself.

To start with, you need to create a start-up file. A Backbase web application uses the *Single Page Interface* (SPI) model. SPI is a data-driven request-response model; each modification of the user interface requires a *data request* instead of a *page request*, res-

©2004-2006 Backbase B.V., All Rights Reserved.

ulting in smaller updates and rendering those updates in the existing page (each user interaction results in a *partial* update).

To develop your first Backbase application and use the examples in the user manual:

1. Create your start-up file, for example `index.html`, to serve as your SPI. You can use the `C:/Backbase/3_1_4/skeletons/basic-start-up.html` code as a basis. If you copy the file in the same directory, you can use it directly. Otherwise, you need to change the `src` attribute in the `script` tag, and the `onload` attribute in the `body` tag, to specify the location of the BPC relative to the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
    <head>
        <title>Backbase startup page</title>
        <script type="text/javascript" src="../bpc/boot.js"></script>
    </head>
    <body onload="bpc.boot('../');">
        <xmp b:backbase="true">
            <div b:action="alert" b:value="You have a working Backbase installation">
                Click here to see if your Backbase installation is working</div>
        </xmp>
    </body>
</html>
```

2. Cut-and-paste the examples between the `xmp` tags in your start-up file. The code between the `xmp` is parsed and rendered by the BPC engine. It marks what is known as the *BXML Space [p. 39]*.
3. Test your code by running the file from your file system.
   To run the file from a web server, you first need to do some configuration. For more information on configuring your web server with Backbase, see the *Getting Started* PDF in the `C:/Backbase/3_1_4/docs` directory. (The PDF describes how to configure the Apache and IIS web servers, but you can use any web server.)

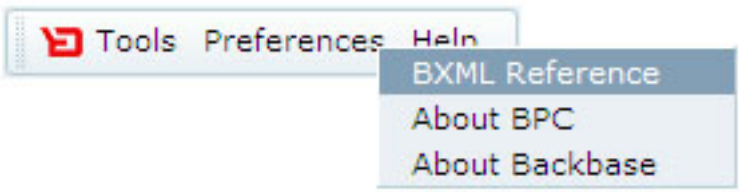
See section **2.3 Booting and Initialization** for an explanation of a startup file.

1.4 Developer Resources

To help you get up to speed developing a Backbase application, the following developer resources are available:

- **Backbase Explorer** - an interactive test area with code samples you can try out and test
- **Starter Kits** - sample Backbase applications designed to help get you started developing more complex web applications
- **User Manual Code Samples** - you can cut-and-paste the examples provided in this manual into a startup file and test.
- **BXML Reference** - detailed reference information on the Backbase `b:` and `s:` tags, commands and events, XPath and Backbase JavaScript functions, is available as a PDF in the `C:/Backbase/3_1_4/docs` directory, or as Online Help (see Backbase Tools).
- **Backbase Tools** - runtime developer tools and BXML reference documentation are available. When you have a running Backbase application, press the **Esc** button to display the Backbase **Tools** menu.

Figure 1.13 - BXML Reference Online Help



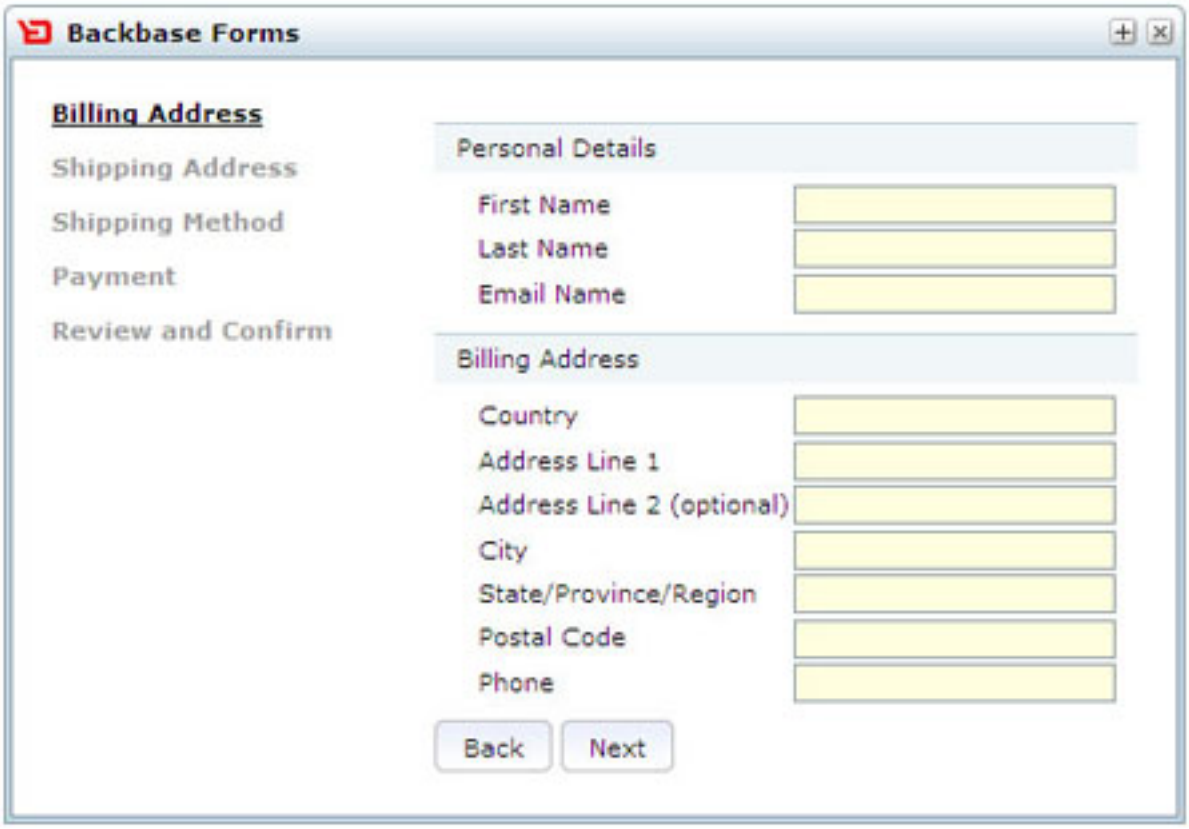
1.4.1 Starter Kits

©2004-2006 Backbase B.V., All Rights Reserved.

The Backbase starter kits that are sample applications designed to help get you started developing more complex web applications. You can use the source code provided by the starter kits as a basis for developing a Backbase Web application. The starter kits are sample applications that demonstrate widely used types of Web applications.

The starter kits are located folder in your `Backbase/3_1_4/starterkits` directory, and documentation for them is provided in the `Backbase/3_1_4/docs` directory.

Figure 1.14 - Forms Starter Kit



©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.15 - Windows Starter Kit

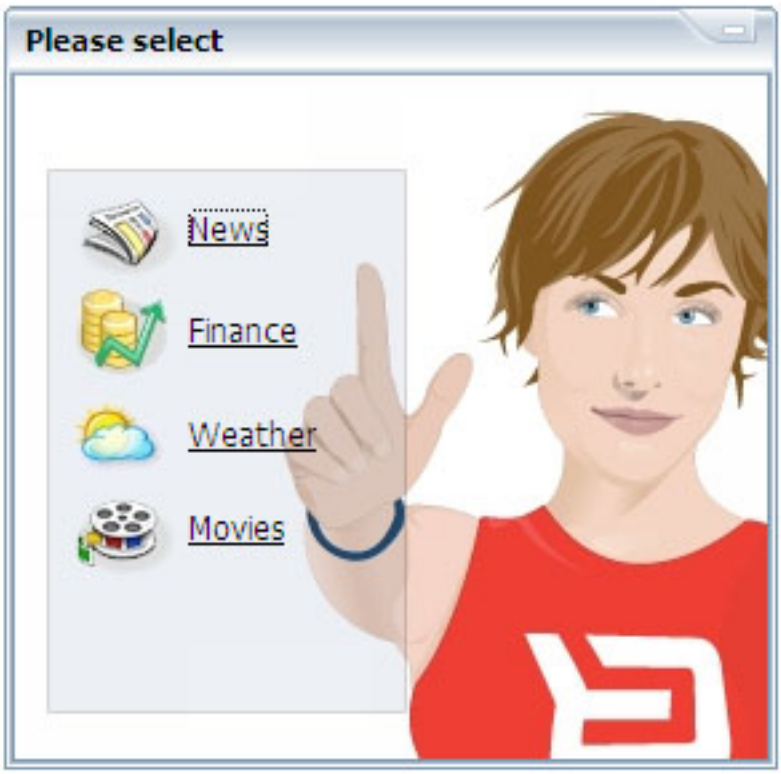
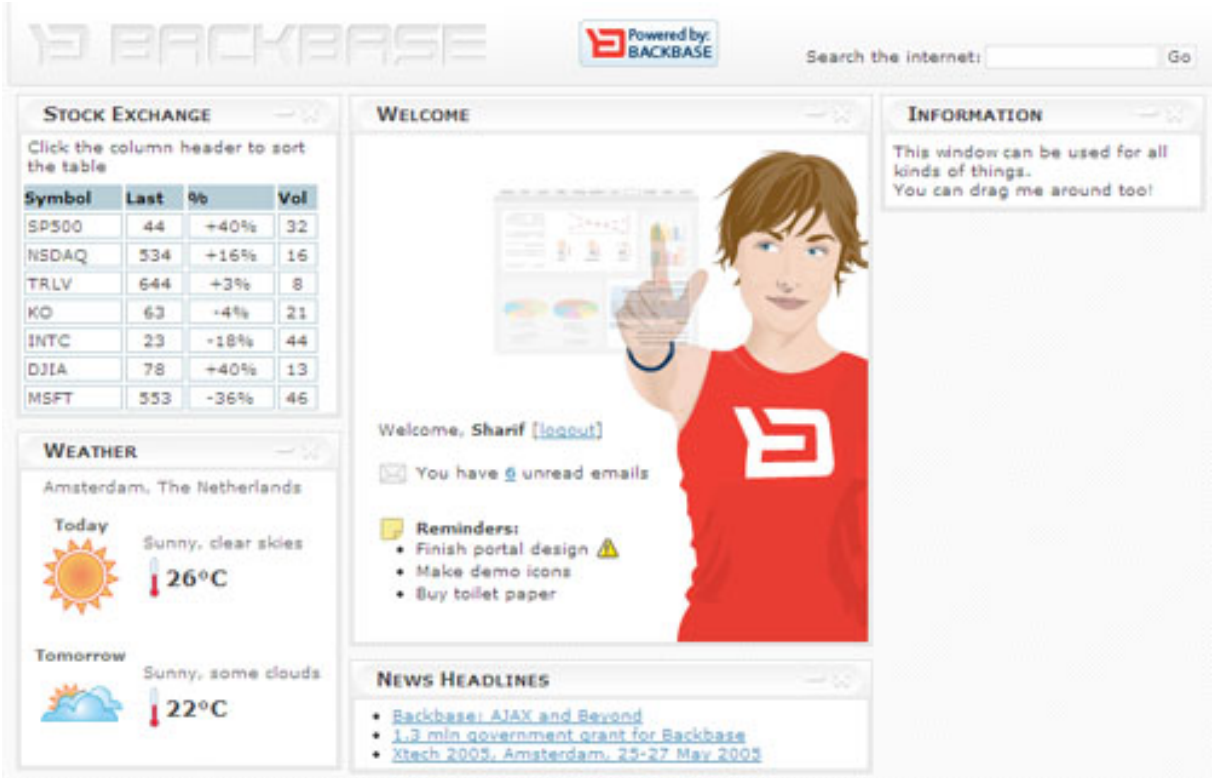
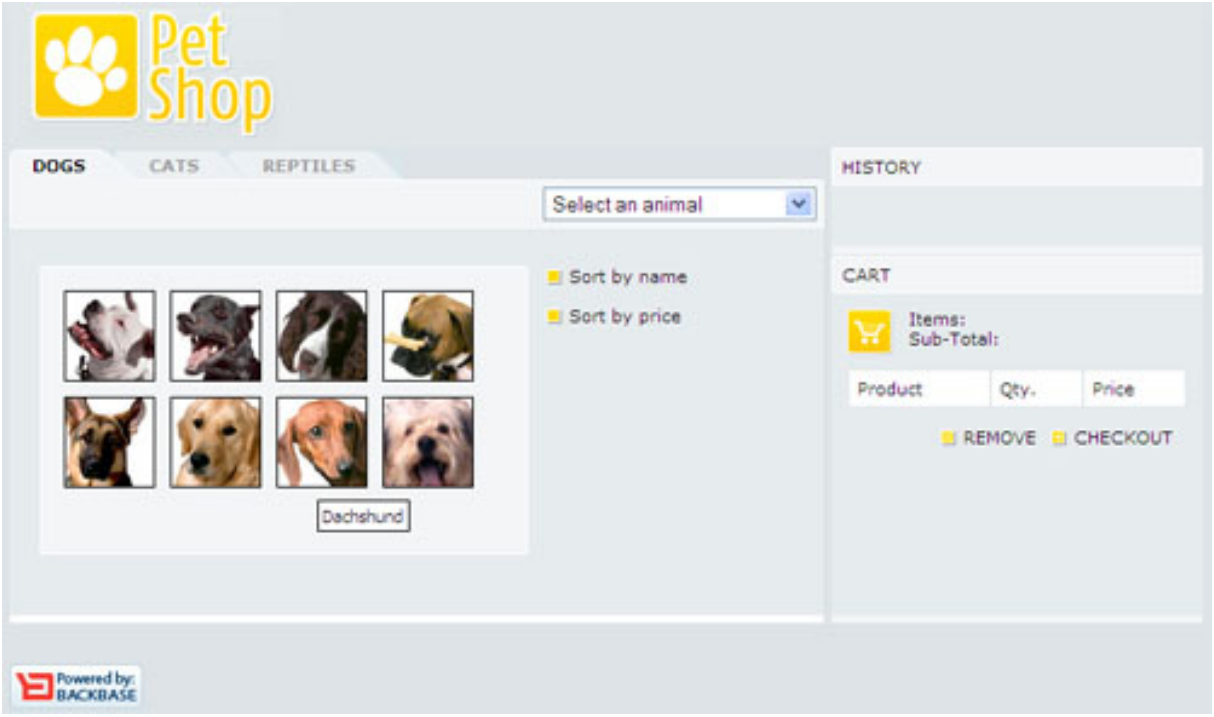


Figure 1.16 - Portal Starter Kit



©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.17 - Petshop Starter Kit



1.4.2 Backbase Explorer

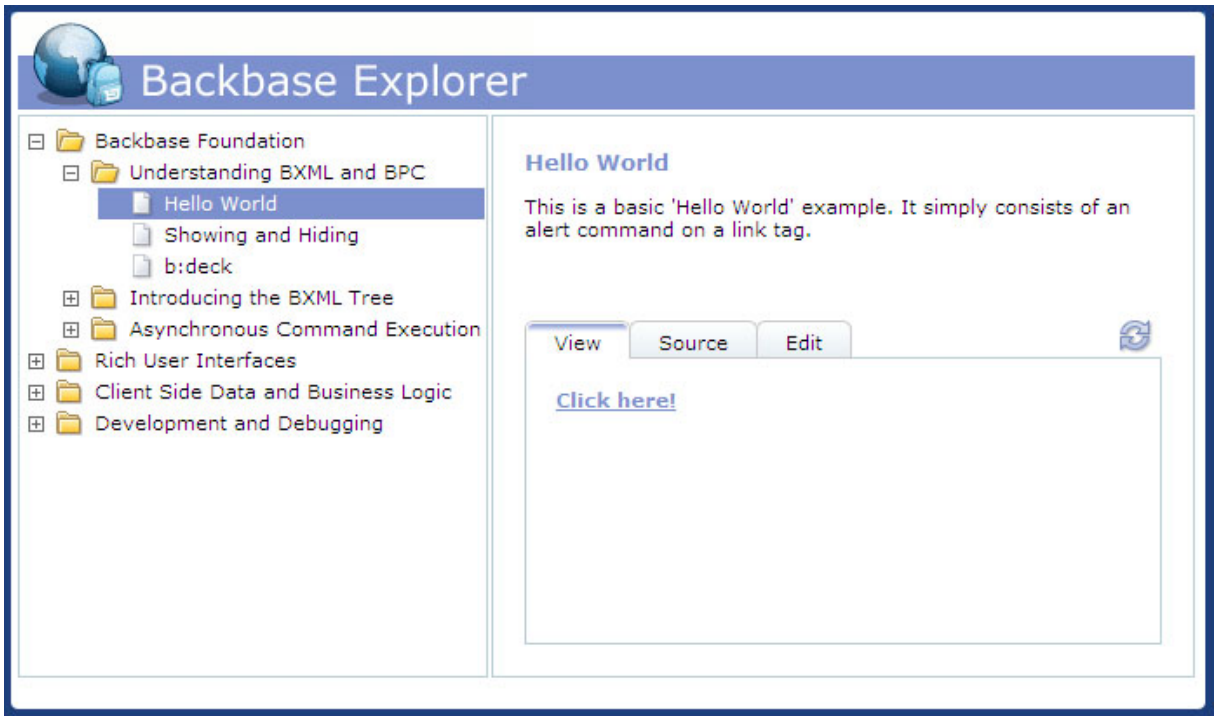
The Backbase Explorer shows samples of the concepts explained in the Backbase User Manual. The structure of the explorer is identical to that of this manual, so you can easily cross-reference the resources. The explorer is an interactive user interface where you can view and try out sample code, and add your own code. You can access the Backbase Explorer from the following URL:

`http://www.backbase.com/demos/explorer/`

The Backbase Explorer is also delivered as part of your Backbase installation, in your `Backbase/3_1_4/explorer` directory.

©2004-2006 Backbase B.V., All Rights Reserved.

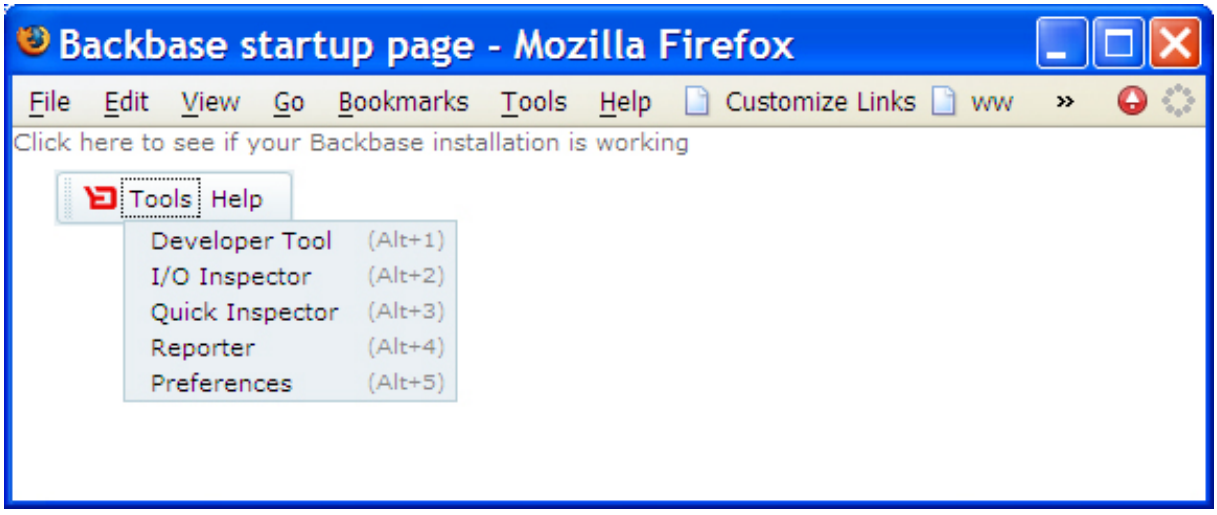
Figure 1.18 - Backbase Explorer



1.5 Runtime Developer Tools

Backbase has runtime developer tools that provide detailed information your runtime Backbase web application. You can access the developer tools via the Backbase Tools menu, which is available in the browser at runtime by pressing the **Esc** key. Pressing the **Esc** shows and hides the menu. The following image shows the Tools menu in a browser window:

Figure 1.19 - Backbase Tools


Backbase startup page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help Customize Links ww
Click here to see if your Backbase installation is working
Tools Help
Developer Tool (Alt+1)
I/O Inspector (Alt+2)
Quick Inspector (Alt+3)
Reporter (Alt+4)
Preferences (Alt+5)

The following table provides an overview of the available tools:

Tools	Description
Developer Tool	Shows a selected element in the document tree and provides detailed information on that element, such as its HTML or BXML source code, its DOM attributes, how the

©2004-2006 Backbase B.V., All Rights Reserved.

Tools	Description
	element is rendered by the browser layout manager, the behaviors it uses, and finally the variables it has at its disposal. The **Developer Tool** also has a **Runtime Tracer** which monitors actions executed at runtime, a **Search Results** tab in which you can view the results of an XPath search, and the **Execute BXML** tab in which you can add and run BXML code on the selected element. Keyboard Shortcut is **Alt+1**.
I/O Inspector	Monitors files that are sent and received and provides its request and response details. Keyboard Shortcut is **Alt+2**.
Quick Inspector	Allows you to get information on a BXML element by moving the mouse over it. Keyboard Shortcut is **Alt+3**.
Reporters	Creates an overview of the building blocks used in the construction of your web application; Behaviors, Ids, TaskLists, Gfx-sets, and Key-Bindings. Keyboard Shortcut is **Alt+4**.
Preferences	Specify Backbase Tools Preferences. Keyboard Shortcut is **Alt+5**.

1.5.1 Developer Tool

The **Developer Tool** provides detailed information on a selected element in your document. It comprises the following panes:

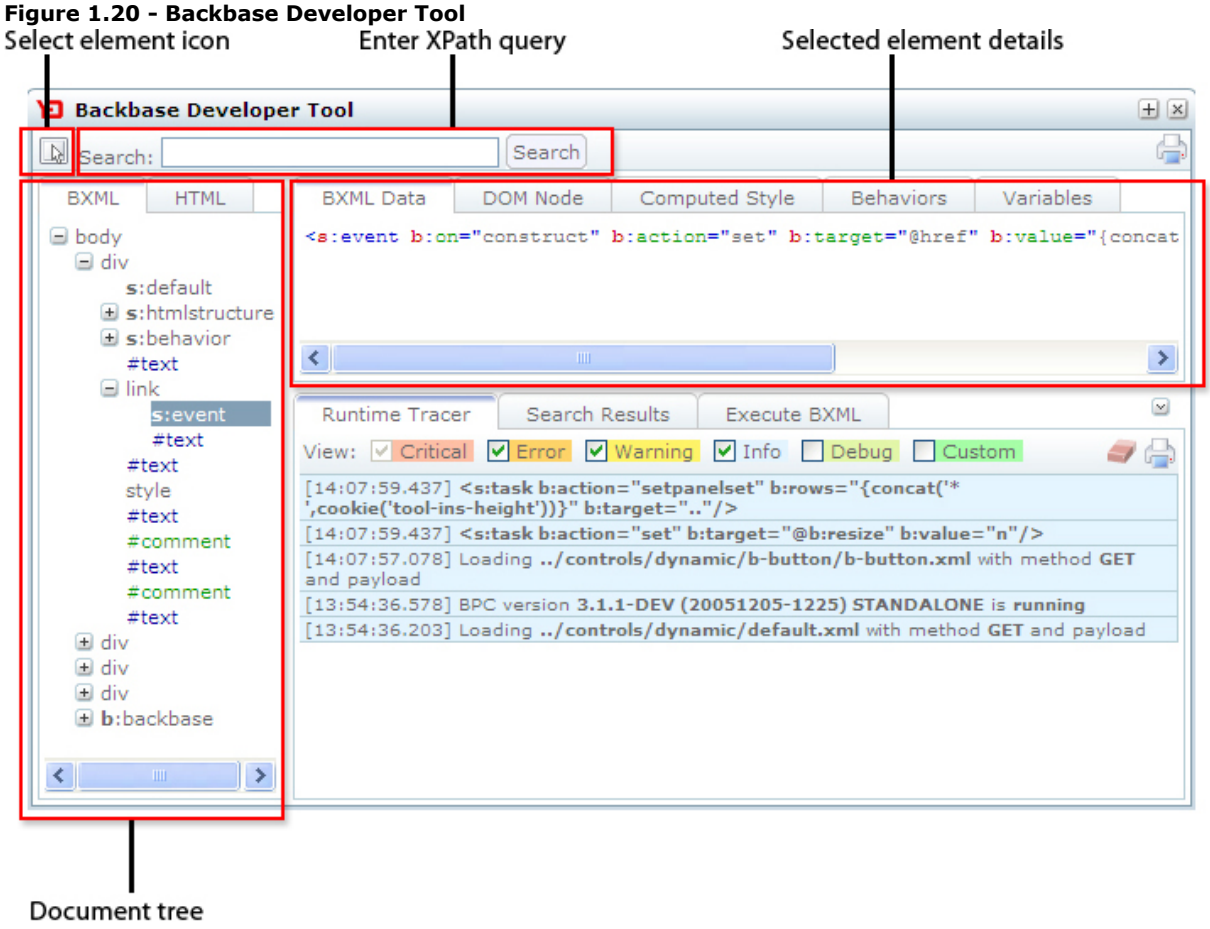
- **Search Bar** - you can select an element for which you want to view its runtime details by clicking the select icon (in the top left-hand corner) and then clicking the element in the browser. Alternatively, you can enter an XPath query to search the document and select the element from the **Search Results** tab. The query has the scope of the current selected element, which is the `body` tag if no element is selected.
- **Document Tree** - you can view a selected element in the **BXML** or **HTML** tab, depending on whether the element is part of the BXML space or HTML space, in the left-hand side pane.
- **Element Details** - in the right-hand side pane, you can view the following detailed information on the element selected in the document tree (the **BXML** or **HTML** tab):
  - **BXML Data** tab shows the BXML code defined for the element
  - **HTML** tab shows the XHTML code defined for the element
  - **DOM Node** tab shows DOM information such as attributes and values,
  - **Computed Style** tab shows the style properties that have been computed by the client browser layout manager to render the object. These properties differ for each layout manager.
  - **Behaviors** tab shows the behaviors used by the element

©2004-2006 Backbase B.V., All Rights Reserved.

    - **Variables** tab shows the variables the element has at its disposal
- In the bottom right-hand pane, the **Backbase Developer Tool** also has:
    - **Runtime Tracer** tab that monitors and logs the execution of your runtime code, and displays by default **Critical**, **Error**, **Warning**, and **Info** messages. You can add the `debug="true"` attribute on an element or behavior to log the actions that are executed on it, and also create custom messages. For more information, see *Creating Custom Messages [p. 27]*.
    - **Search Results** tab in which you can view the results of an XPath search
    - **Execute BXML** tab in which you can add and execute BXML code on the selected element.

If you want to monitor all the actions executed in your application, set `debug="true"` on the `body` element.

Figure 1.20 - Backbase Developer Tool



The following describes how to use the **Developer Tools** based on a simple scenario:

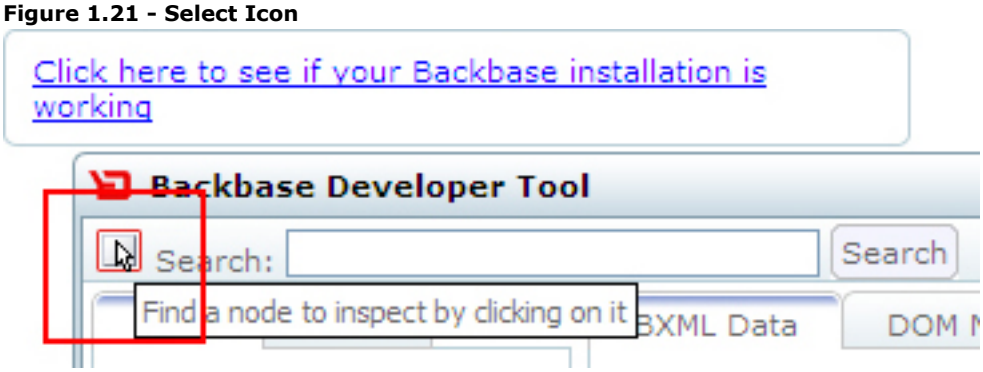
1. Create a start-up file and add the following code:

```
<s:behavior b:name="alert">
    <s:event b:on="command">
        <s:task b:action="alert" b:value="You have a working Backbase installation" />
    </s:event>
</s:behavior>
<b:box>
    <a b:behavior="alert">Click here to see if your Backbase installation is
working</a>
</b:box>
```

2. Run the application in your browser.
3. Press **Esc** to open the Backbase **Tools**, and choose **Tools > Developer Tools**. Alternatively, you can press **Alt+1**.
4. Click the search icon in the top left-hand corner to select it (a red border indicates it is selected). You can now select any element in the browser window.

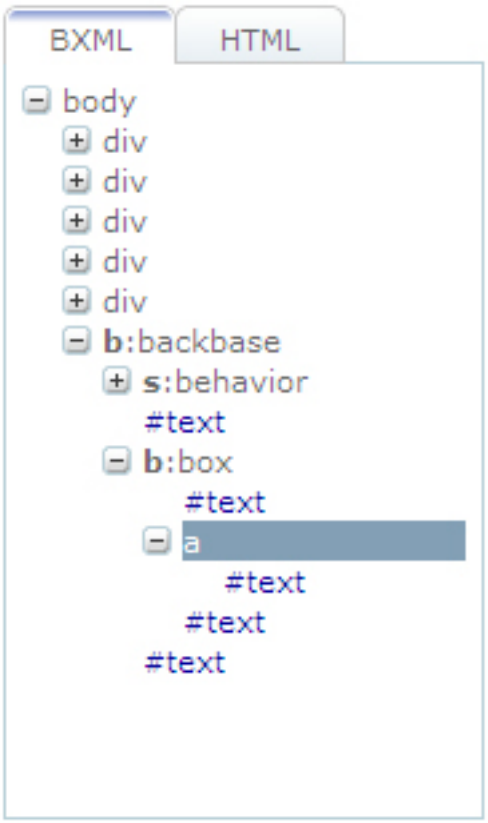
©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.21 - Select Icon



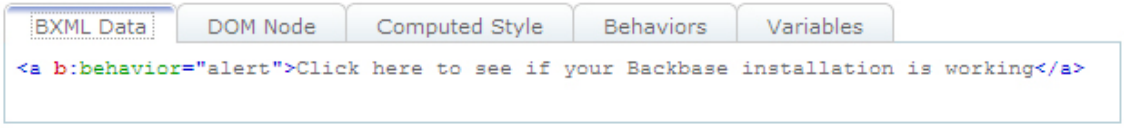
5. Click the link (the a element) in the browser. The element is selected in the **BXML** tab that shows the BXML tree:

Figure 1.22 - BXML Tab



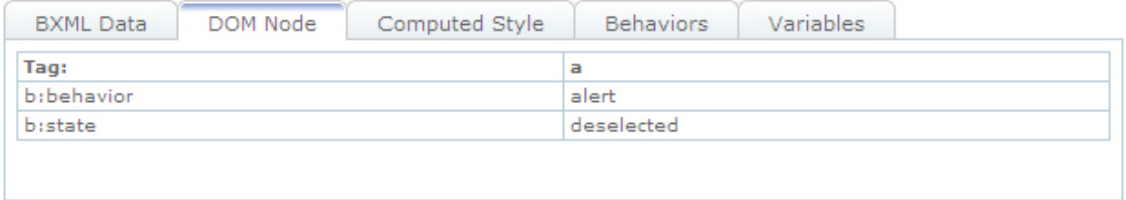
6. In the **BXML Data** tab, view the BXML code defined for the selected element:

Figure 1.23 - BXML Data Tab



7. Click the **DOM Node** tab to view the DOM defined for the selected element:

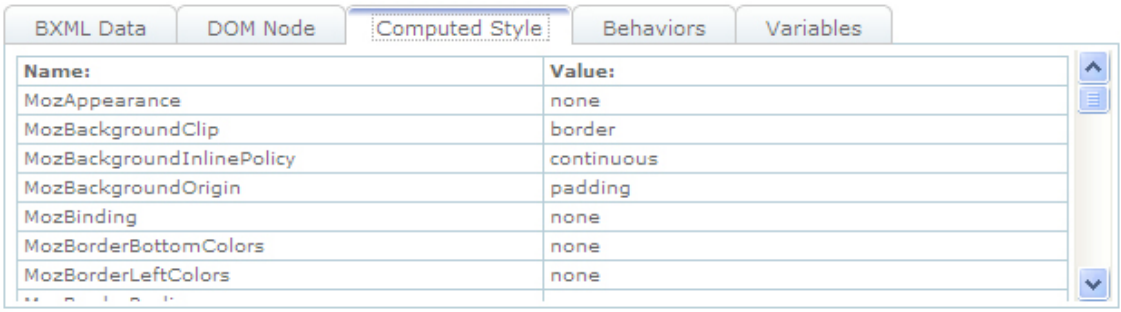
Figure 1.24 - DOM Node



©2004-2006 Backbase B.V., All Rights Reserved.

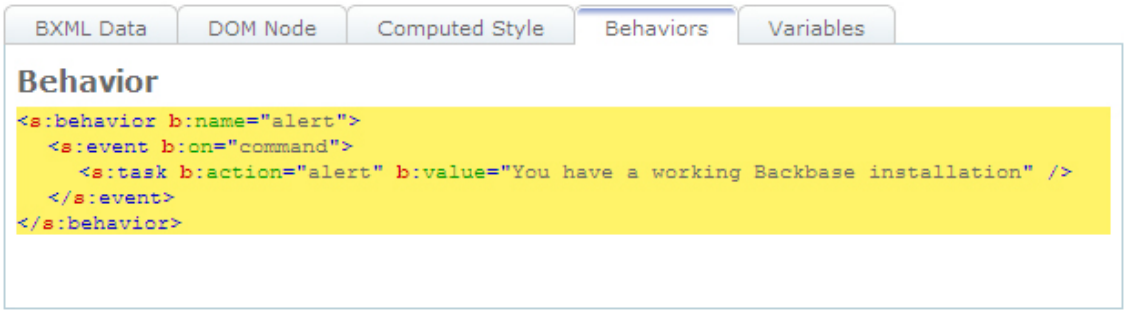
8. Click the **Computed Style** tab to view the style properties that have been computed by the client browser layout manager, for the selected element, to render the object. These properties differ for each browser layout manager.

Figure 1.25 - Computed Style Tab



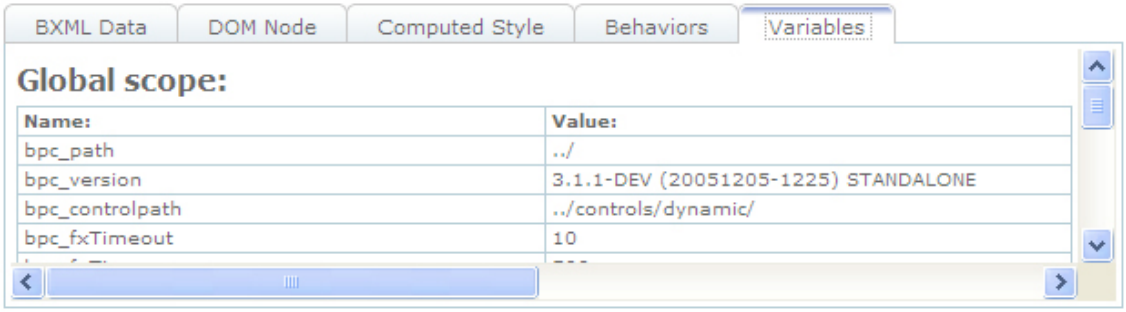
9. Click the **Behaviors** tab to view the behaviors used by the selected element:

Figure 1.26 - Behaviors Tab



10. Click the **Variables** tab to view the variables that are defined and accessible for the selected element:

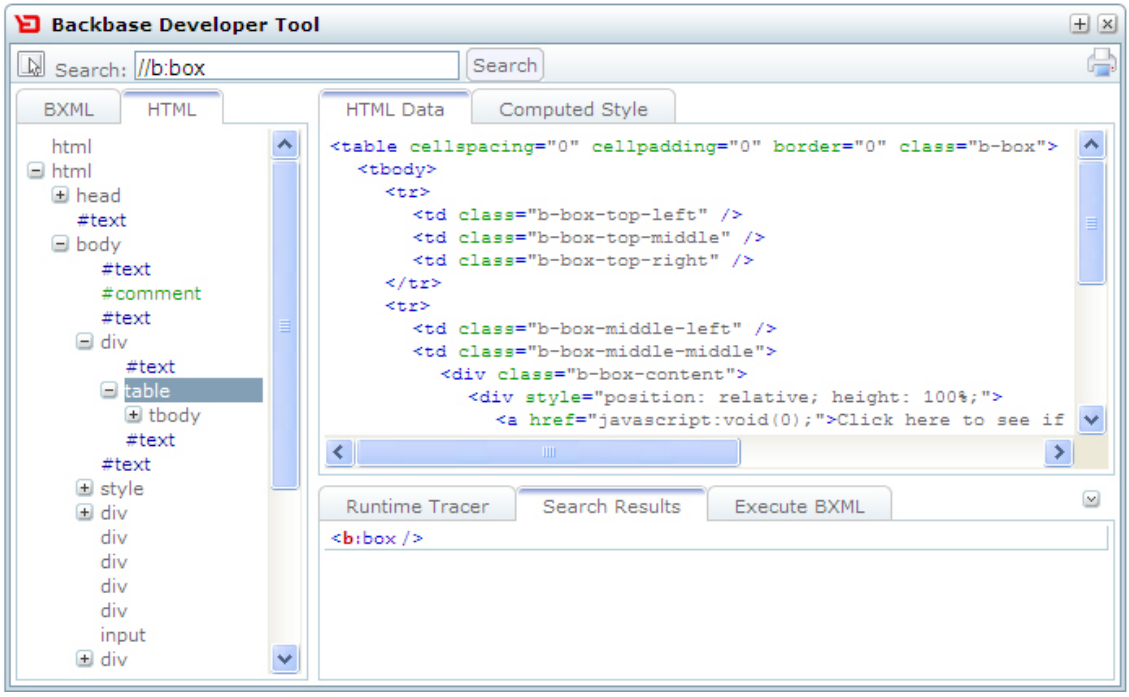
Figure 1.27 - Variables Tab



11. Perform a search of your document by entering an XPath expression, for example:
    - Enter `//b:box` and press **Search**.
    - In the **Search Results** tab, click the link and switch to the **HTML** tab to view the **HTML Data** for the selected element:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.28 - HTML Tree

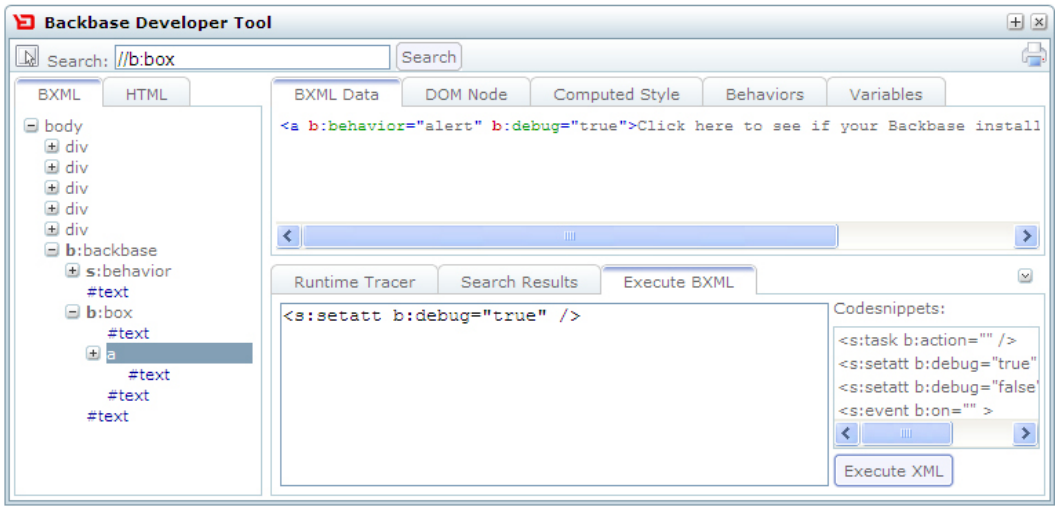


12. Click the **Execute BXML** tab. The **Code snippet** pane provides skeleton code that you can double-click to add into the **Execute BXML** tab. For example, add the following code and click **Execute BXML** to run the code (the code is run on the selected node):
    - `<s:setatt b:style="background: red" />`
    - `<s:task b:action="alert" b:value="Hi!" />`

    BXML is executed on an element selected in the BXML tree.

13. View actions executed in the **Runtime Tracer**:
    - In the **BXML** tab, select the link element (`a`).
    - Open the **BXML Tab**.
    - In the **Execute BXML** tab, enter `<s:setatt b:debug="true" />` and click **Execute BXML**. In the **BXML Tab**, note the attribute has been set.

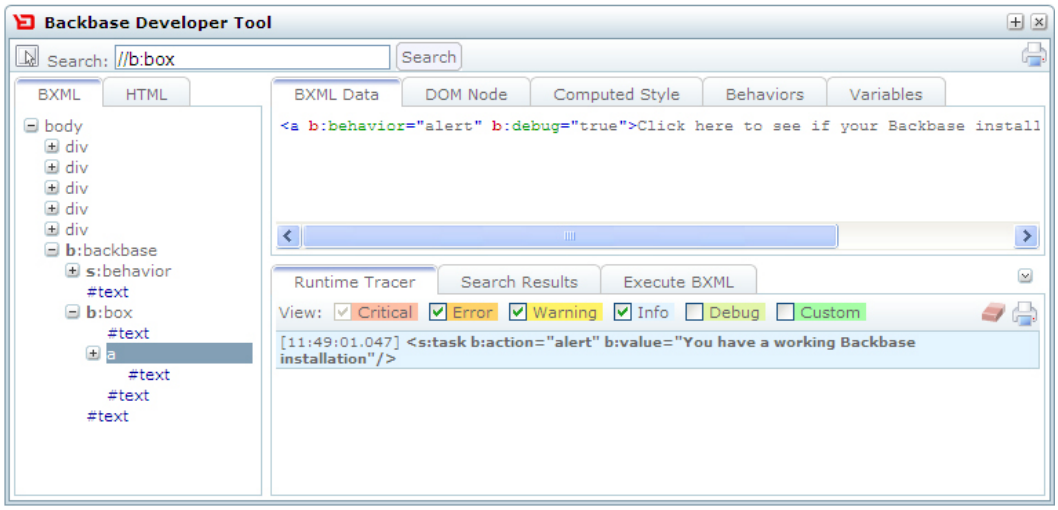
    Figure 1.29 - Execute BXML



    - Open the **Runtime Tracer**.
    - In the browser, click the link. Note the action that has been executed and logged in the **Runtime Tracer**.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.30 - Runtime Tracer



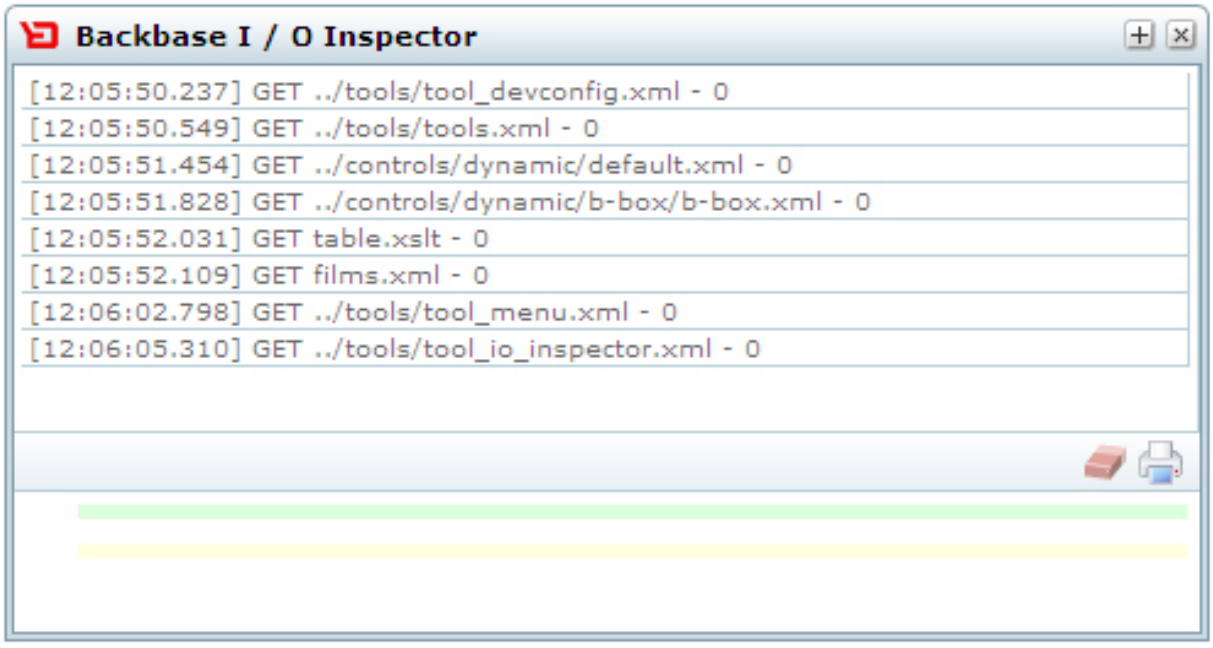
- You can double-click an action in the **Runtime Tracer** to jump to the element on which the action is triggered in the **BXML** tree.

You can refresh the tool detail using the keys **shift+r**

1.5.2 I/O Inspector

The **I/O Inspector** monitors files that are sent and received. Click a file to view its request and response details.

Figure 1.31 - I/O Inspector

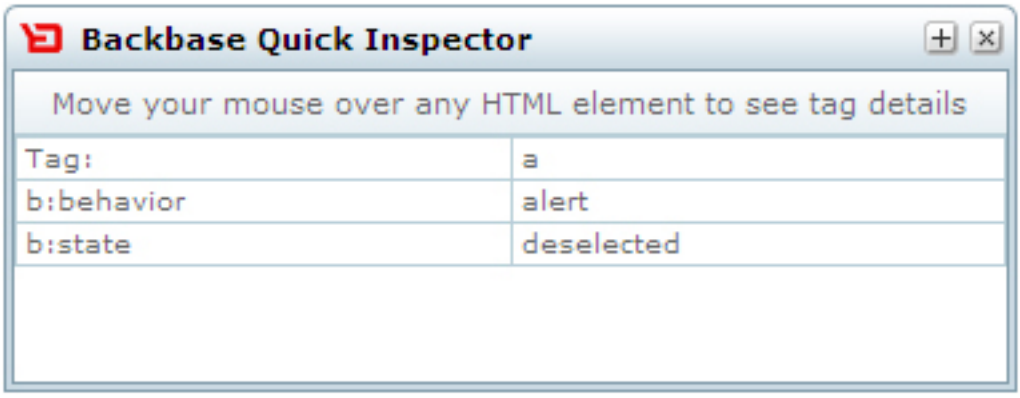

Backbase I / O Inspector
[12:05:50.237] GET ../tools/tool_devconfig.xml - 0
[12:05:50.549] GET ../tools/tools.xml - 0
[12:05:51.454] GET ../controls/dynamic/default.xml - 0
[12:05:51.828] GET ../controls/dynamic/b-box/b-box.xml - 0
[12:05:52.031] GET table.xslt - 0
[12:05:52.109] GET films.xml - 0
[12:06:02.798] GET ../tools/tool_menu.xml - 0
[12:06:05.310] GET ../tools/tool_io_inspector.xml - 0

1.5.3 Quick Inspector

The **Quick Inspector** allows you to get information on a BXML element over which the mouse is currently hovering. Open the **Quick Inspector** and move the mouse (or any other pointing device) over an element to get more information on the element's attributes.

©2004-2006 Backbase B.V., All Rights Reserved.

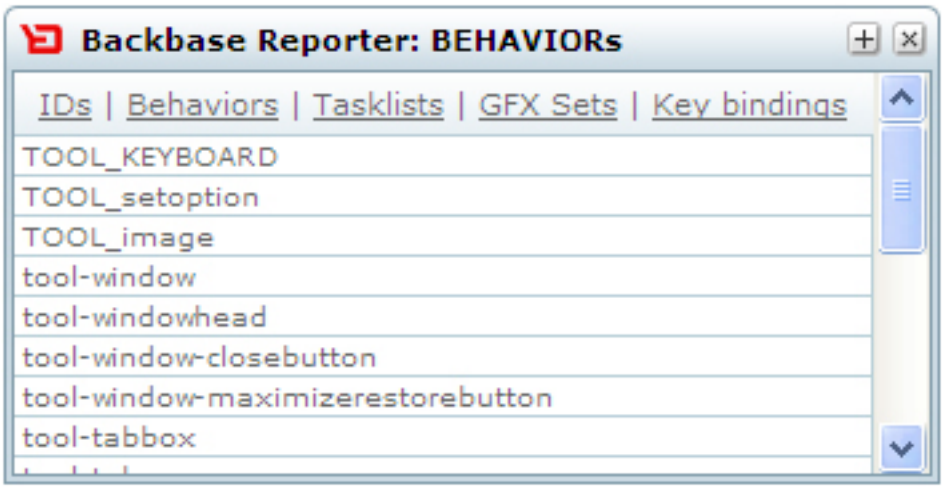
Figure 1.32 - Quick Inspector



1.5.4 Reporters

Creates an overview of the building blocks used in the construction of your web application; Behaviors, Ids, TaskLists, Gfxsets, and Key-Bindings.

Figure 1.33 - Reporters



The following table provides an overview of the reporters available:

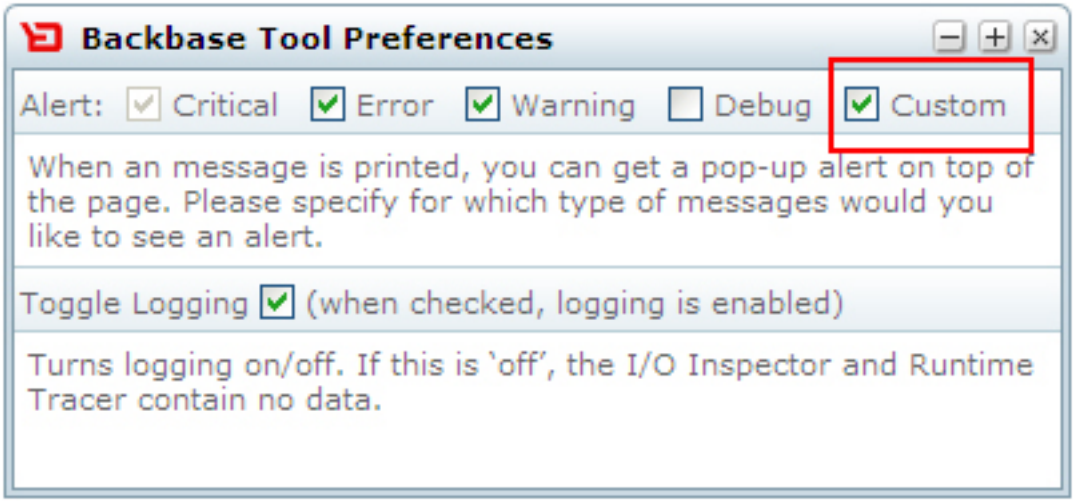
Reporters	Description
Behaviors	Creates an overview of all currently loaded behaviors (remember that states and events are a part of behaviors).
Ids	Creates an overview of all currently loaded `ids`.
TaskLists	Creates an overview of all currently loaded `tasklists`. A tasklist is a container for a sequence of commands that are executed when the tasklist is called.
Gfxsets	Shows images used for a widget (a graphic set can only be defined for the slider and the tree control)
Key-Bindings	Shows an overview of all the keys attached to commands.

©2004-2006 Backbase B.V., All Rights Reserved.

1.5.5 Preferences

Specify Backbase Tools Preferences.

Figure 1.34 - Preferences


Backbase Tool Preferences
Alert: Critical Error Warning Debug Custom
When an message is printed, you can get a pop-up alert on top of the page. Please specify for which type of messages would you like to see an alert.
Toggle Logging (when checked, logging is enabled)
Turns logging on/off. If this is 'off', the I/O Inspector and Runtime Tracer contain no data.

1.5.6 Excluding Backbase Tools

The Backbase Tools are always loaded by default. You can exclude the tools by adding the `b:devconfig="false"` attribute to the HTML `body` tag as follows:

Example 1.3 - Excluding Backbase Tools

```
<body onload="bpc.boot('/bpc/3_1_4/');" b:devconfig="false">      ...</body>
```

You also need to set the `b:devconfig` attribute if you have defined a custom development configuration file (`tool_devconfig.xml`).

1.5.7 Configuring Backbase Tools

The `tool_devconfig.xml` file is the configuration file of the Backbase Tools. You can modify the file to change predefined keyboard shortcuts, if your web application already uses keys that conflict with these shortcuts, and to localize the messages that appear in the **Runtime Tracer**.

1. Open the `tool_devconfig.xml` in the `/tools` directory of your Backbase installation in a text editor.
2. Edit the following section of the file to change the keyboard shortcuts:

```
<s:behavior b:name="TOOL_KEYBOARD">
    <s:whenactive>
        <s:keys b:keys="27" b:action="tool-toggle" b:target="menu" />
        <s:keys b:keys="alt+49" b:action="tool-toggle" b:target="inspector" />
        <s:keys b:keys="alt+50" b:action="tool-toggle" b:target="io_inspector" />
        <s:keys b:keys="alt+51" b:action="tool-toggle" b:target="g_inspector" />
        <s:keys b:keys="alt+52" b:action="tool-toggle" b:target="reporter"
b:report="default" />
        <s:keys b:keys="alt+53" b:action="tool-toggle" b:target="settings" />
        <s:keys b:keys="alt+54" b:action="tool-help" />
        <s:keys b:keys="shift+r" b:action="tool-detail-refresh" />
    </s:whenactive>
</s:behavior>
```

The value of the `b:keys` attribute can be the key character (case is ignored), or the decimal ASCII value of the key. For example, `27` is the **Esc** key. For more information on defining keyboard shortcuts, see *Capturing Keys [p. 93]*.

3. If you change the location of your configuration file, you must specify the location in the `b:devconfig` attribute on the `body` tag on your startup file. For example:

©2004-2006 Backbase B.V., All Rights Reserved.

```
<body onload="bpc.boot('/bpc/3_1_4/');" b:devconfig="myRootDir/tool_devconfig.xml">
     ...</body>
```

1.5.8 Creating Custom Messages

In Backbase you can add custom messages to your code using the `b:action="msg"` command. You can view your custom messages in the alertbar, or add them to the Runtime Tracer.

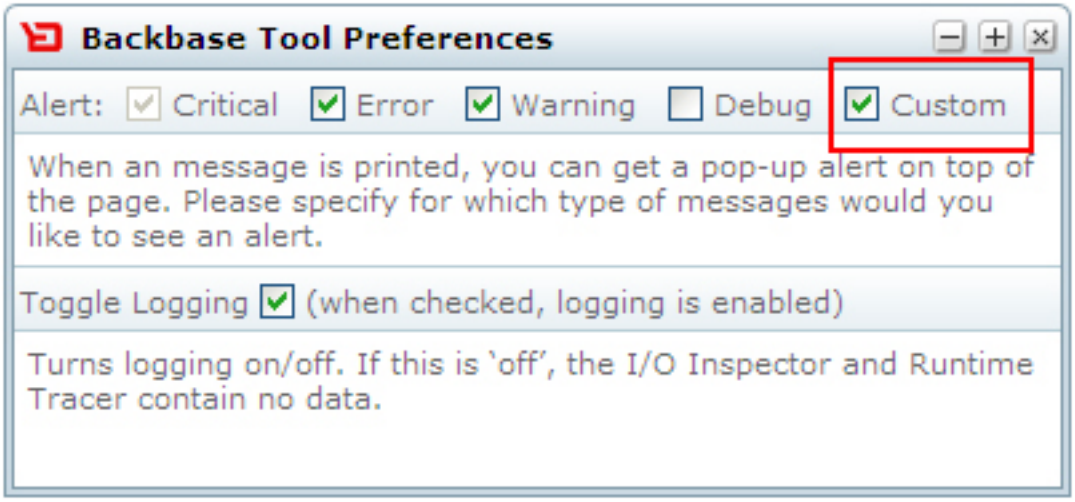
In the following example, you create a custom message that creates the message `...generating a custom message`, and then change the tools preferences and runtime tracer settings to have the message displayed:

1. Create a start-up file and add the following code:

```
<a>Click here to generate a custom message...
    <s:event b:on="command">
        <s:task b:action="msg" b:value="...generating a custom message" />
    </s:event>
</a>
```

2. Run the application in your browser.
3. Press **Esc** to open the Backbase **Tools**.
4. Choose **Tools > Preferences**.
5. In the **Backbase Tool Preferences** dialog, select the **Custom** checkbox.

Figure 1.35 - Backbase Tool Preferences


Backbase Tool Preferences
Alert:
Critical
Error
Warning
Debug
Custom
When an message is printed, you can get a pop-up alert on top of the page. Please specify for which type of messages would you like to see an alert.
Toggle Logging
(when checked, logging is enabled)
Turns logging on/off. If this is 'off', the I/O Inspector and Runtime Tracer contain no data.

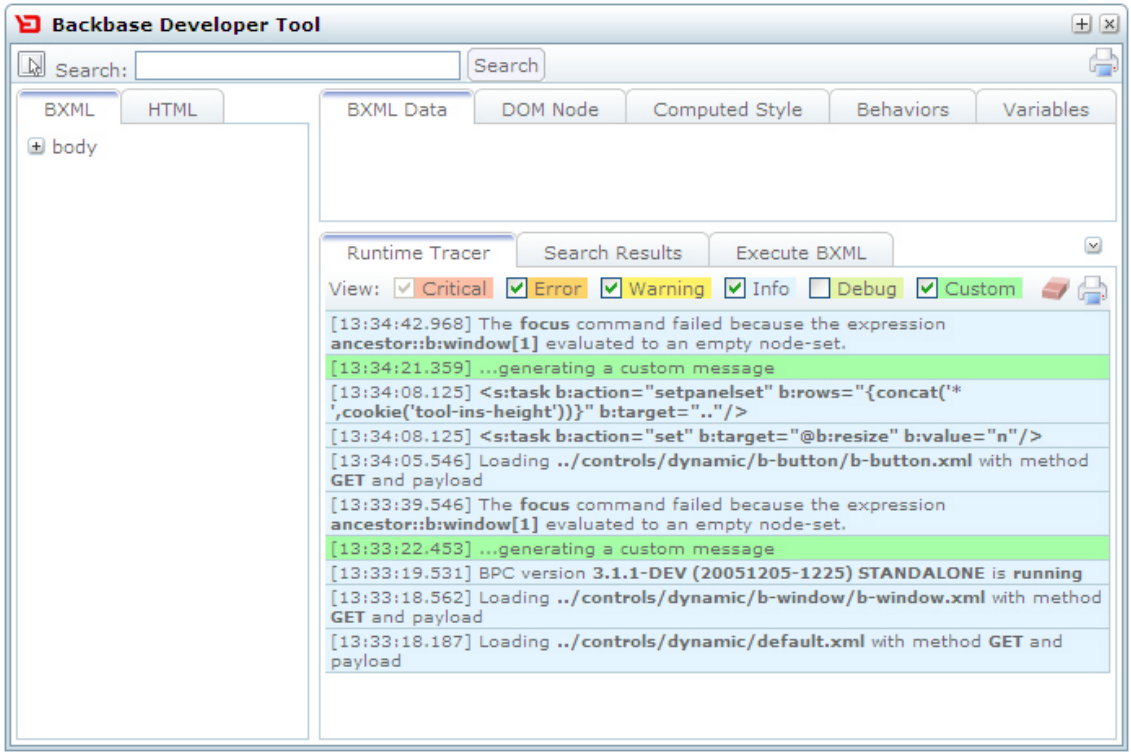
6. Click the link to generate the custom message. The message appears in the alertbar:

Figure 1.36 - Backbase Alertbar

7. Choose **Tools > Developer Tool**.
8. Open the **Runtime Tracer** tab and select the **Custom** checkbox as follows:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 1.37 - Backbase Runtime Tracer



©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 2 - Understanding BXML and BPC

2.1 BXML Overview

BXML stands for **B**ackbase e**X**tensible **M**arkup **L**anguage. BXML is a declarative, XML-based language for developing Rich Internet Applications (RIA). The BXML vocabulary is an extension of XHTML that provides tags for declaring rich and highly interactive web sites. You can freely mix BXML tags and attributes with HTML. The BXML tags are defined in the following namespaces:

- `xmlns:b=http://www.backbase.com/b` for visual tags and attributes (`b:` prefixed tags)
- `xmlns:s=http://www.backbase.com/s` for systems tags (`s:` prefixed tags)

The information in this section is intended to provide you with an overview of the BXML language. The sections in the rest of the manual goes into detail about the most important BXML tags. To start with, the BXML language is categorized as follows:

- **System tags (s:tags)** - for defining event handlers, behaviors, fxstyles, your own controls, and so on.
- **Backbase controls (b:tags)** - for out-of-the box user interface controls, such as menus, tabs, trees, sliders, interface controls (custom client controls)
- **Generic attributes (b:tags)** - define functionality for programming, GUI functionality, tree manipulation, and focus model that you can apply to all XHTML and BXML elements
- **Events and commands (b:tags)** - for defining the events that are triggered on elements, and commands that define the actions that take place when the event is triggered

The following code is a very simple example to demonstrate the key facets of BXML. In the example, the user interface consists of a `b:box` control with an XHTML link element defined within it. The box control is a simple layout control that provides default styling and layout for its contents (any BXML or XHTML). The first three lines of code define an event handler for the `command` event:

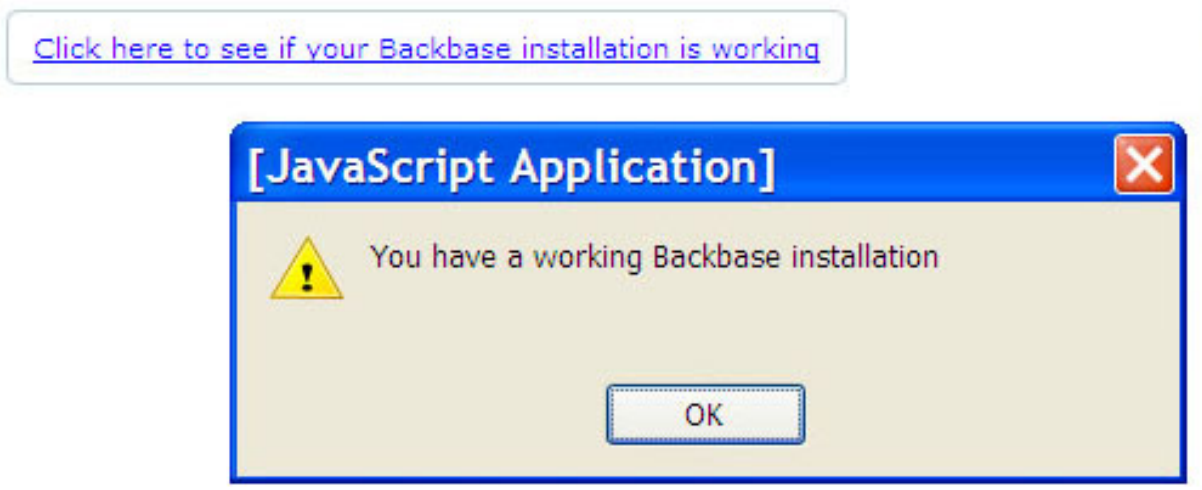
Example 2.1 - Simple BXML Example

```
<b:box>
    <a>Click here to see if your Backbase installation is working
        <s:event b:on="command">
            <s:task b:action="alert" b:value="You have a working Backbase installation" />
        </s:event>
    </a>
</b:box>
```

When the `b:on="command"` event is triggered on the link element, that is when the user clicks on the link, the event handler is triggered; an alert dialog box appears with the specified value:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 2.1 - Basic BXML Example


Click here to see if your Backbase installation is working
[JavaScript Application]
You have a working Backbase installation
OK

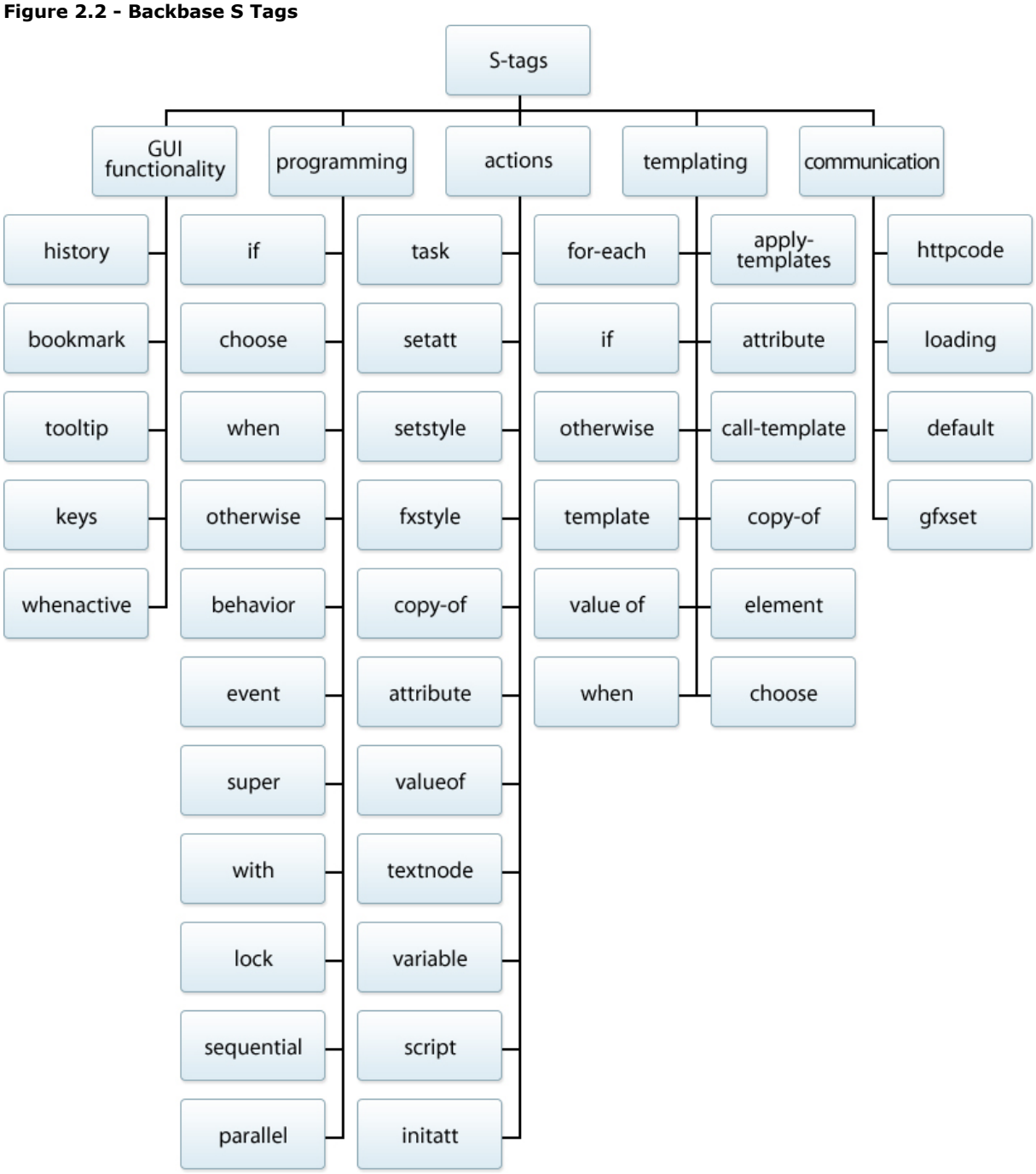
If no event is defined on an element, the `command` event is the default event that is triggered. In the above example therefore, writing `<b:box><a>Click here to see if your Backbase installation is working</a></b:box>` is equivalent.

2.1.1 System Tags (s:tags)

`s:` prefixed tags are used for system tags. Each **s:** tag has its own specific attributes. The following illustration provides an overview of the `s:` tags:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 2.2 - Backbase S Tags

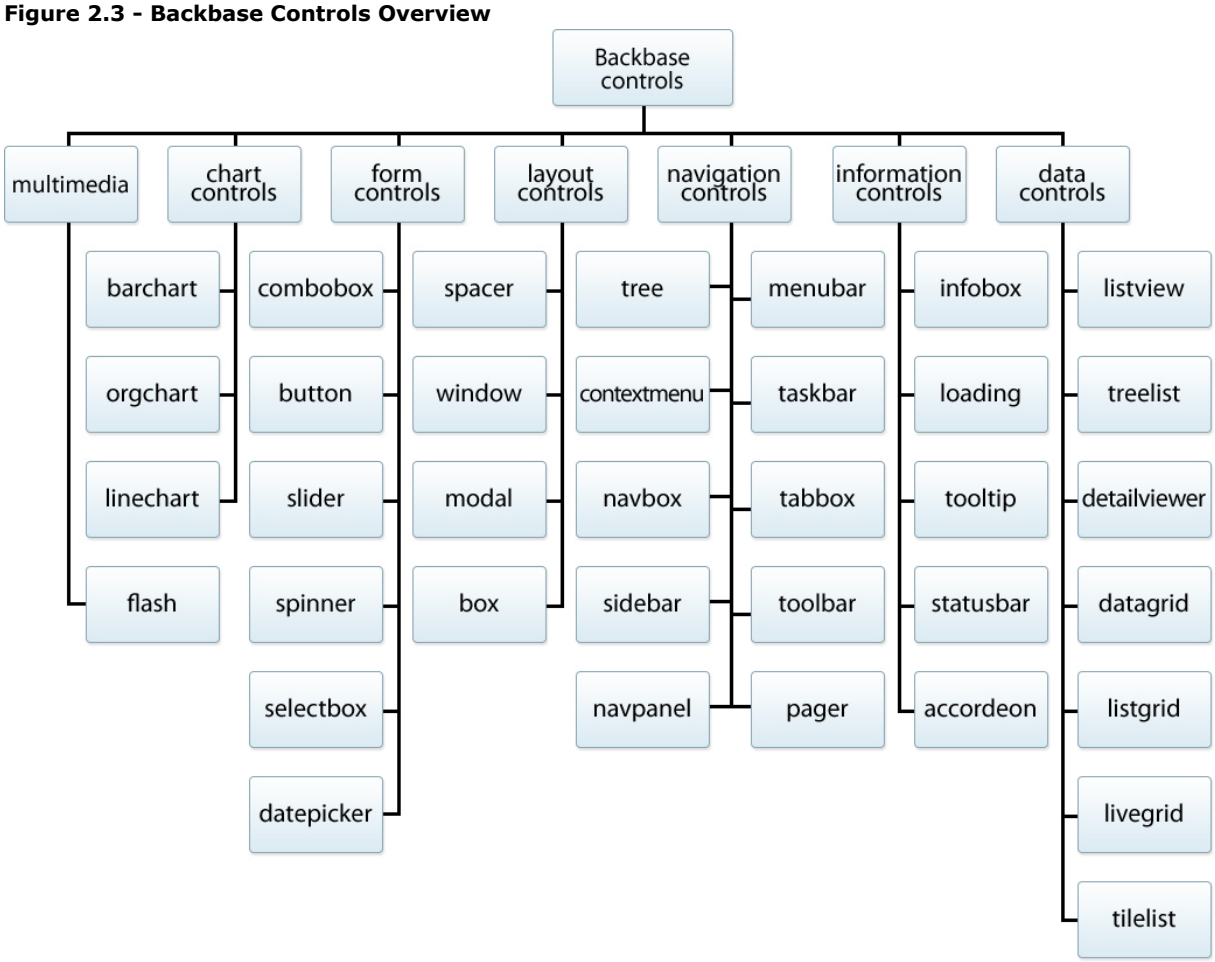


2.1.2 Backbase Controls

Backbase controls are the visual interface controls that are defined in the `b:` namespace. Each **b:** tag has its own specific attributes, and you can also apply generic attributes to these elements. The following illustration provides an overview of the Backbase out-of-the-box controls:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 2.3 - Backbase Controls Overview

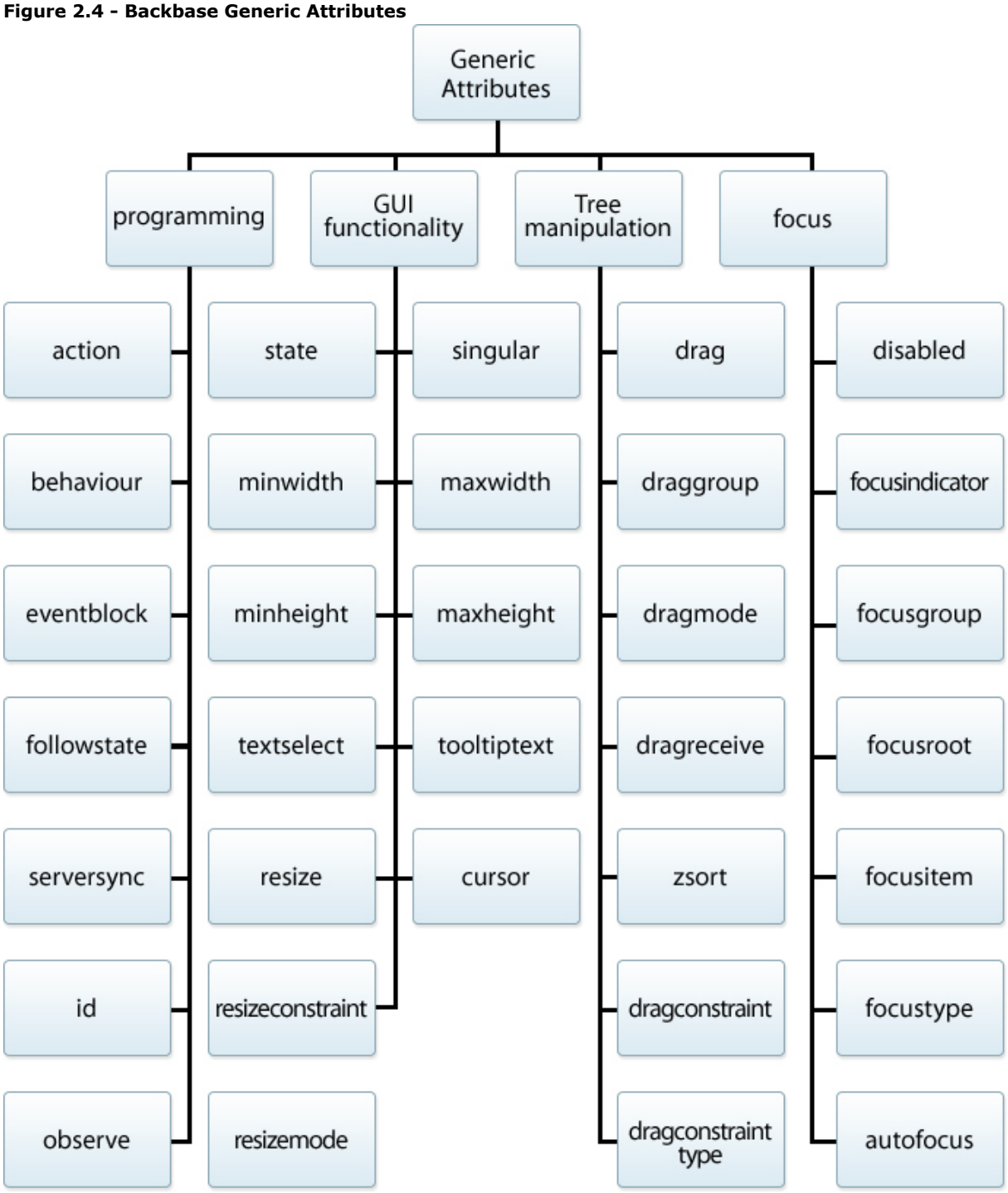


2.1.3 Generic Attributes

Generic attributes define functionality for programming, GUI functionality, tree manipulation, and the focus model. You can also apply generic attributes to almost any BXML or XHTML. The following illustration provides an overview of the generic attributes:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 2.4 - Backbase Generic Attributes



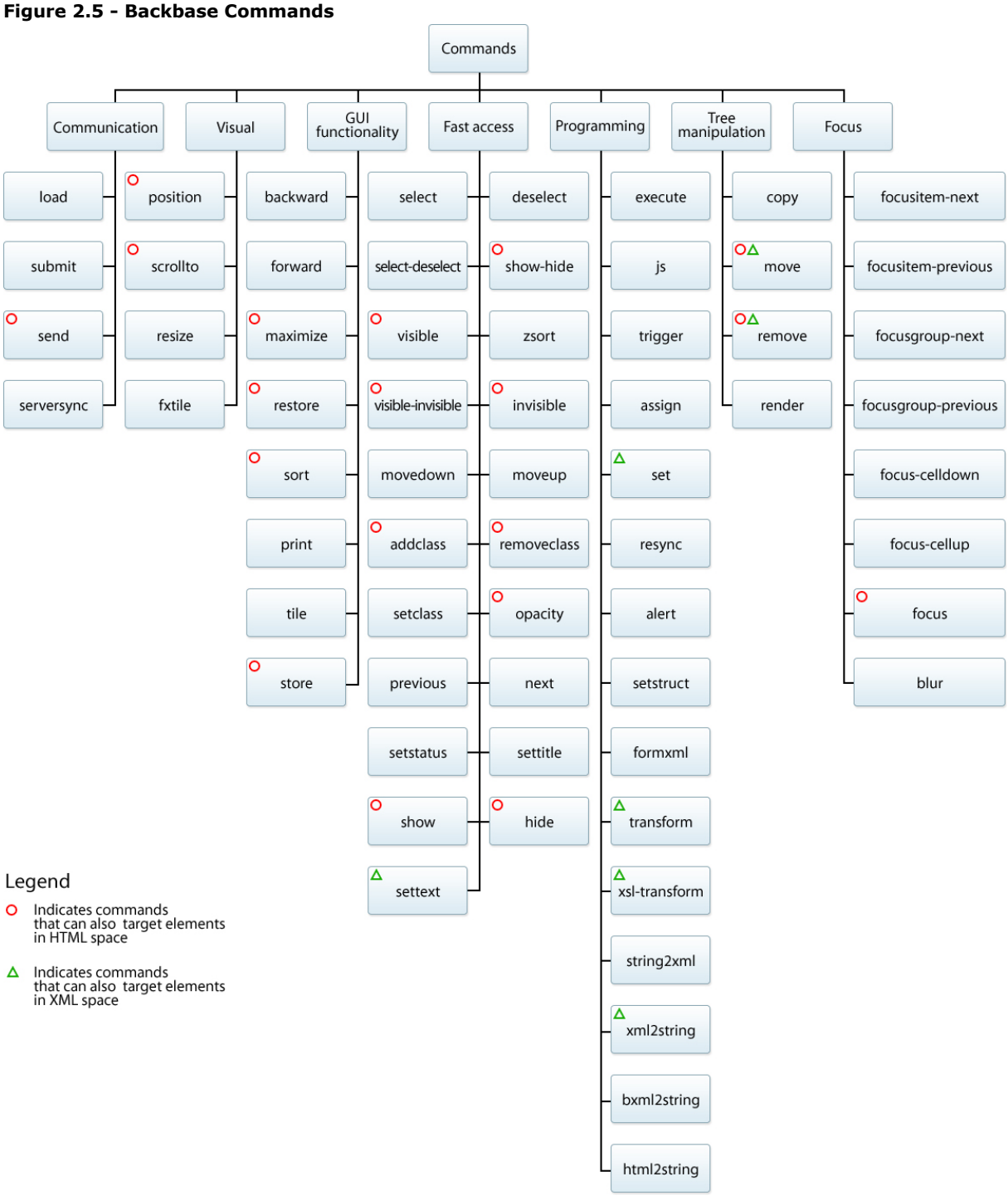
There are also configuration attributes, `b:controlpath` and `b:skinbase`. For more information, see *Setting the Control Path [p. 84]*.

2.1.4 Events and Commands

Events add interactivity to your SPI. Most events are *User Events [p. 89]* that are triggered when the user does something, such as clicking a button, but there are also *System Events [p. 91]* that are not directly caused by the user, for example the `construct` event fires when a page is constructed. When an event takes place, you can define the actions, or *commands*, that should take place by defining an *event handler* for the event. The following illustration provides an overview of the Backbase commands:

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 2.5 - Backbase Commands



In Backbase, you can attach an event to any element in the UI. You then create an event handler to determine what takes places when the event is triggered, for example what should happen when a user clicks on a link. An event handler is a container for executing commands. The event handler handles the event by executing a command, or series of commands.

For more information on event handling, see *Defining Event Handlers [p. 87]*

There are also widget-specific commands: `bufferdirty`, `datepicker`, and `setpanelset`.

©2004-2006 Backbase B.V., All Rights Reserved.

2.2 Backbase Presentation Client

The Backbase Presentation Client is a robust and generic GUI rendering engine, which provides a declarative client-side runtime for developing AJAX Rich Internet Applications. The BPC operates in all main browsers, is ultra light and is fully compatible with Web Standards like XHTML, CSS, JavaScript, XML, and XPath: this set of technologies is commonly referred to as AJAX. The Backbase Presentation Client runs in most browsers.

Once activated within the web browser, the BPC creates a client-side presentation framework, which includes facilities for state management, event management, asynchronous parallel processing, client-server synchronization and incremental data loading. This framework utilizes the processing power of the local machine to directly execute the web application's interaction and presentation logic and allows for the instantiation of Single Page Interfaces (SPIs). Although the BPC is developed in JavaScript, you do not have to know JavaScript to work with Backbase: you can simply add BXML tags to your (X)HTML pages.

2.3 Booting and Initialization

The startup file is the starting point of your web application and is located in the root of your container's web application folder. The startup file must contain the syntax to load the BPC, and the external resources that contain the content and define the style of your web application.

The following is an example of a simple SPI startup file:

Example 2.2 - Startup File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
    <head>
        <title>Backbase startup page</title>
        <script type="text/javascript" src="../bpc/boot.js"></script>
    </head>
    <body onload="bpc.boot('../');" b:controlpath="../controls/backbase/" b:skinbase="none">
        <xmp b:backbase="true">
            <div b:action="alert" b:value="You have a working Backbase installation">
                Click here to see if your Backbase installation is working</div>
        </xmp>
    </body>
</html>
```

The following table describes the elements of the startup file:

Elements	Description
`!DOCTYPE definition`	The XHTML standard defines three Document Type Definitions. It is recommended that you use XHTML 1.0 Strict DTD.
Namespaces	In the `html` tag, enter the XHTML namespace and the BXML namespace declarations for `b:` and `s:` tags.
`<script>`	In the `head` section of the startup file, define the script to handle the variables for booting the BPC. In this example, the `/bpc` is a relative path.
`body`	`onload="bpc.boot('../');"` loads the BPC with a relative path to the `boot.js`.

©2004-2006 Backbase B.V., All Rights Reserved.

Elements	Description
	The `body` tag runs the `bpc.boot` method when the document has been loaded. The method has a `('../')` parameter which is used to indicate the path of the BPC. `b:controlpath="../controls/backbase/"` indicates the path to the controls you want to use, `b:skinbase` the `default.xml` file. For more information, see *Setting the Control Path [p. 84]*.
`xmp`	The `xmp` tag marks the BXML space. The `b:backbase="true"` attribute indicates all BXML `s:` and `b:` tags must be parsed and rendered by the BPC. Outside the `xmp` tags, you can use standard HTML and browser markup. For more information, see *BXML Space [p. 39]*

©2004-2006 Backbase B.V., All Rights Reserved.

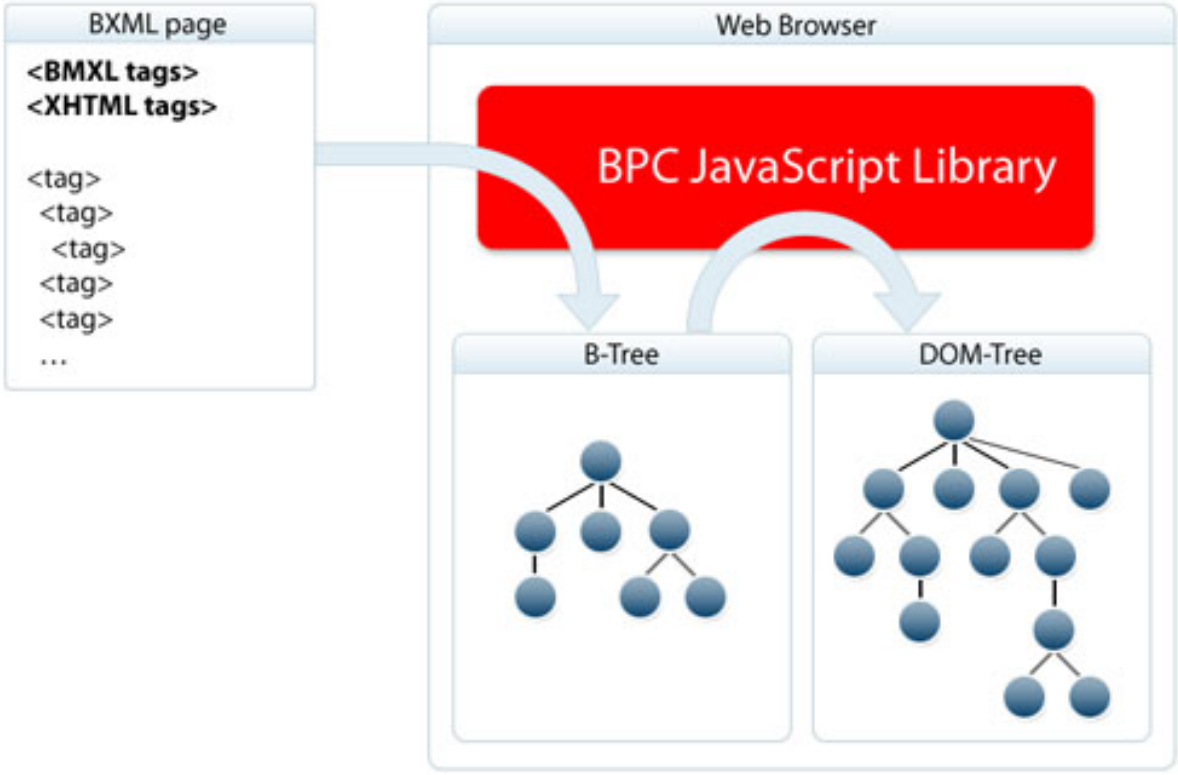
Chapter 3 - Introducing the BXML Tree

3.1 Managing the BXML Tree and DOM Tree

Understanding the *BXML Tree* and *DOM Tree* is fundamental to understanding how you can create dynamic rich internet applications (RIA). The objective of the BXML Tree and the DOM Tree is the same: by defining a logical structure of documents, and the way a document is accessed and manipulated, it allows you to dynamically access and update the content, structure and style of documents:

- The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. In the DOM, documents have a logical structure which is very much like that of a tree, hence the term DOM Tree.
- The BXML Tree is the object representation of BXML, defined within `xmp` tags. At runtime, the Backbase Presentation Client (BPC) first parses the BXML and creates the BXML Tree. It then translates the BXML Tree into a standard DOM Tree that is rendered by the browser.

Figure 3.1 - Tree Manipulation



In Backbase, you can manipulate the objects in the BXML Tree using the concept of *targeting*. Targeting allows you to point to, and navigate through objects in the BXML Tree to perform dynamic actions such as copy, move, or remove, and to load new data content. Targeting is performed using XPath.

The DOM Tree is the model understood and managed by the browser; the BXML Tree the model understood and managed by the BPC (the BXML Tree is a tree representation of your BXML source code). The DOM Tree translated from the BXML Tree is the HTML representation of the BXML Tags. Therefore, when you manipulate the BXML Tree, the DOM Tree is also automatically updated to reflect the changed structure.

Note: For more information, see W3C's DOM documentation resources

©2004-2006 Backbase B.V., All Rights Reserved.

[http://www.w3c.org/DOM/DOMTR] .

3.2 Targeting

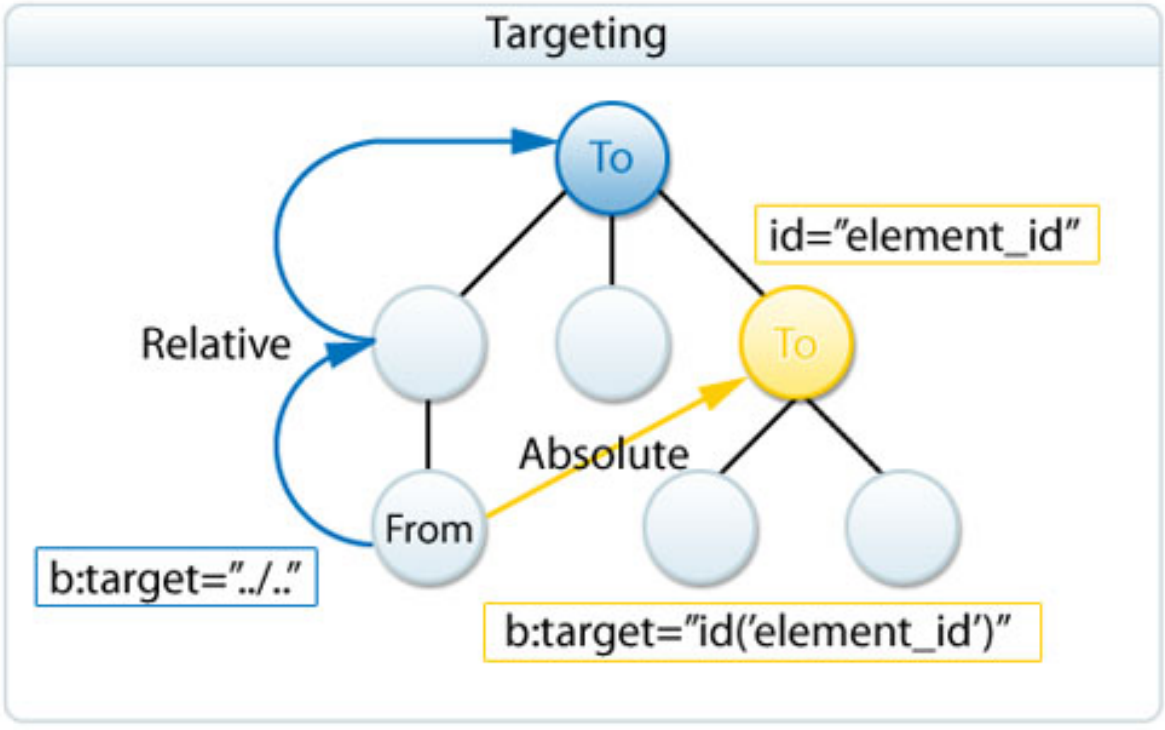
Targeting is one of the core concepts of the SPI model. Targeting is used throughout a Backbase application to locate an object in the BXML Tree using *XPath*, with the objective to dynamically modify the appearance and contents of the user interface. For example, targeting allows you to perform actions such as drag-and-drop, move, copy, and to load new data into an existing page.

You can address nodes in the BXML tree with XPath using *relative* or *absolute* targeting:

- **Relative targeting** - you can use XML navigation (see *XPath Axis [p. 131]*) to address the hierarchical nodes in the tree, for example `b:target="../.."`
- **Absolute targeting** - you can set IDs on elements that you then can reference directly. For example, where the element ID is `id="element_id"`, you can target the element using `b:target="id('element_id')"`

Note: XPath allows you to navigate through elements and attributes in an XML document. XPath expressions return either a node-set, a String, a Boolean, or a number. Backbase supports the W3C *XPath 2.0* standard.
It is important for you to become familiar with XPath and the targeting concept. Many of the examples in the user manual demonstrate targeting. For more information on XPath Axis, see *XPath Axis [p. 131]* and *XPath Examples [p. 136]* or on XPath in general see W3Schools XPath Tutorial [http://www.w3schools.com/xpath/] .

Figure 3.2 - Targetting



The following example is a simple document that uses targeting to show and hide elements:

Example 3.1 - Targeting

```
<b:box>
    <h3>The Seven Samurai (1954)</h3>
    <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
    <p style="display: none" id="sbox-showhide">          The film takes place in war-ridden
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
16th century Japan, where a village of farmers         hire seven ronin (lordless samurai) to
fight for them against a band of marauding robbers. The Seven Samurai is widely regarded as
one of the greatest Japanese films ever made.</p>
    <p b:action="show-hide" b:target="id('sbox-showhide') | a">
        <a style="display:none">&lt;&lt;&lt; Show less</a>
        <a>Read more &gt;&gt;&gt;</a>
    </p>
</b:box>
```

The following image shows the page in an initial state:

Figure 3.3 - Show-Hide: Initial State

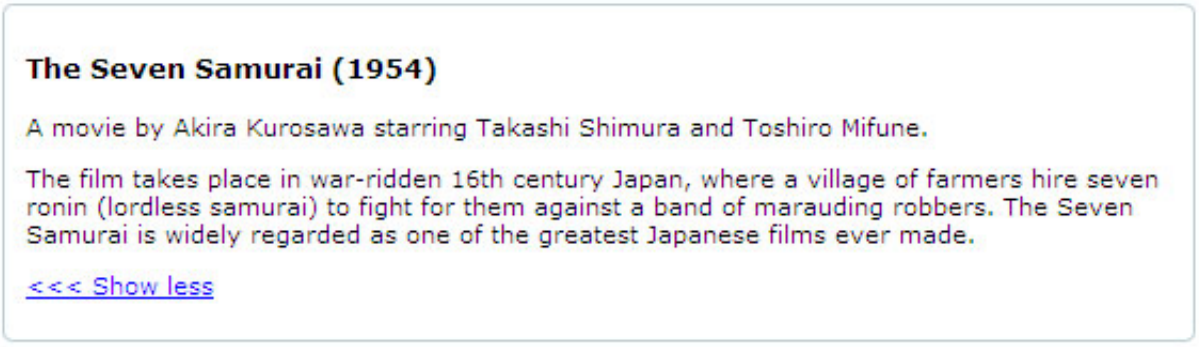
In the example, the following line defines a simple event handler that shows or hides the elements it targets, based on the elements's current *state [p. 103]*.

Example 3.2 - Simple Event Handler

```
<p b:action="show-hide" b:target="id('sbox-showhide') | a" />
```

The `b:target` uses *absolute* targeting to target the element whose id is `sbox-showhide`, and *relative* targeting to target all `a` (link) elements in the context node. The `b:box` is a custom control used for layout purposes, wrapping its contents in a box and applying simple styling.

Figure 3.4 - Show-Hide: Clicked State


The Seven Samurai (1954)
A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.
The film takes place in war-ridden 16th century Japan, where a village of farmers hire seven ronin (lordless samurai) to fight for them against a band of marauding robbers. The Seven Samurai is widely regarded as one of the greatest Japanese films ever made.
<<< Show less

3.2.1 Targeting the HTML Space

A Backbase application is defined within the `xmp` tags of your single page interface (SPI). The `xmp` tags mark the *BXML space*; the BXML and HTML elements that are parsed by the BPC engine and rendered in the BXML Tree. By default, targeting works on tree elements in the BXML space. However, within your Backbase application, you can demarcate areas that belong to the *HTML space* using `b:xhtml` tags; elements within these tags are not parsed by the BPC. Putting elements in the *HTML space* is useful for integration with existing applications, and to improve performance by using XHTML for simple mark-up and simple UI elements (the innercontent of Backbase controls belong to the HTML space; for more information, see *Defining HTML Structures [p. 110]*.

As elements within the `b:xhtml` tags are not parsed by the BPC, you cannot by default

©2004-2006 Backbase B.V., All Rights Reserved.

target these elements because they do not have a node-set representation in the BXML tree. To enable you to target elements in the HTML space, Backbase provides the following custom XPath functions known as *walkers*:

- `html()` - for targeting nodes in the HTML space. The `html()` walker instructs the BPC to go to the representation in the DOM Tree.
- `bxml()` - allows you to switch from the HTML space back to the BXML space

To illustrate the differences between the trees in the HTML and BXML spaces, take the following example; it results in a BXML Tree with three node-sets, and an HTML Tree with five nodes sets. To be able to target the `span` element in the HTML space, the link element event handler uses the `html()` function (the event handler puts the node content into an alert dialog box):

Example 3.3 - HTML Space Example

```
<a b:action="alert" b:value="{following::b:xhtml/html()/div/span}">Click me</a>
<b:xhtml>
    <div>
        <span>HTML Space</span>
        <b:bxml />
    </div>
</b:xhtml>
```

In the following example, clicking the button triggers an event handler that targets elements in the BXML and HTML Spaces to show and hide these elements.

Example 3.4 - Show-Hide HTML Space Example

```
<div style="background-color: #FE987E; width: 400px; height: 100px">These elements are in the
BXML space
    <b:button>Click here to Show-Hide elements in the BXML and HTML Space
        <s:event b:on="command">
            <s:task b:action="show-hide" b:target="id('html_space')/html()/div/span" />
            <s:task b:action="show-hide" b:target="id('bxml_space')/div[1]" />
        </s:event>
    </b:button>
    <div id="bxml_space">
        <div>Clicking the button toggles the display style property of this element in the
BXML Space</div>
    </div>
</div>
<b:xhtml id="html_space">
    <div style="background-color: #80FFB3; width: 400px; height: 100px">These elements are in
the HTML Space
        <span>Clicking the button toggles the display style property of this element in the
HTML Space</span>
    </div>
</b:xhtml>
```

In the following example, the second link targets elements in the HTML space using the `html()` walker, and once in the HTML space it targets elements within `b:bxml` element using the `bxml()` walker:

Example 3.5 - BXML Space Walker

```
<a b:action="alert" b:value="{following::b:xhtml/html()/div/span}">To HTML Space</a>
<span>|</span>
<a b:action="alert" b:value="{following::b:xhtml/html()/bxml()/b:bxml/b:box}">To BXML Space
within HTML Space</a>
<b:xhtml>
    <div>
        <span>This element is in the HTML Space</span>
        <b:bxml>
            <b:box>This box is in the BXML Space</b:box>
        </b:bxml>
    </div>
</b:xhtml>
```

In the HTML space, you must target attributes using the JavaScript conventions. For example, you target the `class` attribute using the syntax: `b:value="{html()/div/@myClassName}"`. (To target the `class` attribute in the BXML space, use the syntax `bxml()/div/@class`.)

©2004-2006 Backbase B.V., All Rights Reserved.

3.2.2 Commands for the HTML Space

You cannot use all the commands available in the BXML language on elements in the HTML space. The following is a list of commands you can use on elements in the HTML space (and BXML space):

- `visible`, `invisible` and `visible-invisible`
- `show`, `hide` and `show-hide`
- `maximize`, `store` and `restore`
- `move` and `remove`
- `addclass` and `removeclass`
- `position`
- `scrollto`
- `send`
- `opacity`
- `focus`
- `sort`

3.2.3 Commands for the XML Space

The following is a list of commands that you can use to target elements in the XML space (raw XML), or that expect XML as input:

- `settext`
- `set`
- `move`
- `remove`
- `transform`
- `xsl-transform`
- `xml2string`

You can also use XPath expressions in attributes of type String to target String values, such as text nodes, in the BXML space, HTML space, and XML space.

3.3 Tree Manipulation

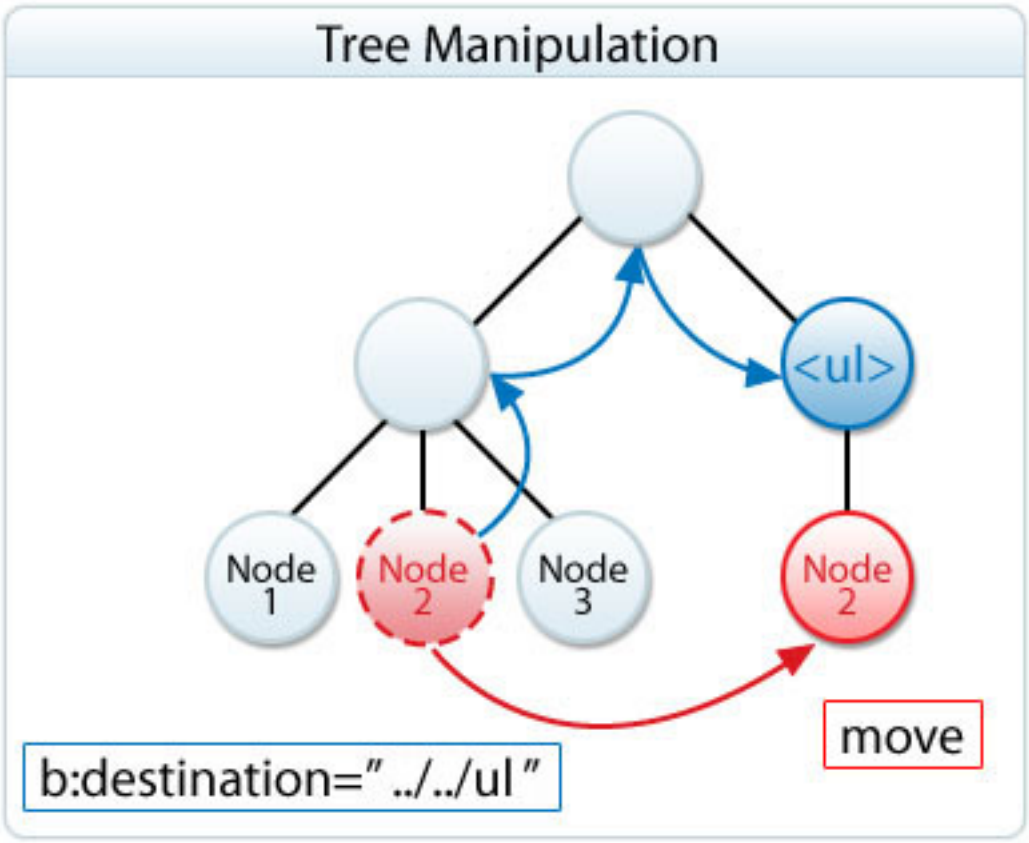
Manipulating the objects in the BXML Tree using XPath to target elements is what makes the user interface interactive and dynamic. This section describes how you can manipulate the BXML Tree to copy, remove, and move elements within the BXML Tree.

Moving takes an element and corresponding child nodes from one location in the BXML Tree and inserts the node set in another location. *Copying* makes a copy of an element at a location in the BXML Tree and inserts the copy of the node set it in another location. *Removing* deletes the element at a location in the BXML Tree.

To be able to manipulate the tree, you need to visualize the BXML SPI document as a tree structure, with familiar grandparent/parent/child/sibling relationship hierarchies of the BXML and XHTML nodes. Targeting elements, and performing actions on them, then becomes relatively simple.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 3.5 - Tree Manipulation



At runtime, the BPC performs BXML tree manipulation on the client-side, without page reloading.

You can perform tree manipulation as follows:

- **Copying Elements** - use the `b:action="copy"` command for copying nodes, or `s:render` for copying, moving, transforming, and adding nodes
- **Moving Elements** - use the `b:action="move"` command for moving nodes (or the `movedown` and `moveup` commands)
- **Removing Elements** - use the `b:action="remove"` command for removing nodes

3.3.1 Copying Elements

The `s:render` tag allows you to copy elements. An element which is copied from one location in the document to another persists in the original location, which makes it possible to create several copies of the same element. The `s:render` tag requires the following attributes:

- `b:destination` - targets the destination (the receiving element) of the copy using an XPath expression
- `b:mode` - specifies where the data is inserted **relative** to the receiving element. The attribute can have the values: `asfirstchild`, `aslastchild`, `before`, `replace`, `replacechildren` (the default if `b:mode` is not specified).

In the following example, each time you click the `span` element which contains the event handler (`s:event`), the contents the `s:render` tag are copied and placed after the element with the id `copyTarget`:

Example 3.6 - Copying Elements

```
<div>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    <span>
        <s:event b:on="command">
            <s:render b:destination="id('copyTarget')" b:mode="aslastchild">
                <div>
                    <b>Backbase Powered Web Applications</b>
                </div>
            </s:render>
        </s:event>
Click Here To Copy
    </span>
    <div id="copyTarget"></div>
</div>
```

`s:render` tags can only be direct child elements of `s:event`, `s:tasklist` and `s:execute` tags.

You can also use the `copy` command that copies an element `b:source` attribute to a `b:destination`. Again, the `b:mode` attribute specifies where the data is inserted relative to the target destination.

3.3.2 Moving Elements

You can move elements and corresponding child nodes using the `move`, `movedown` and `moveup` commands (the latter two commands amount to XPath shorthand expressions). The `move` commands require the following attributes:

- `b:destination` - targets the destination (the receiving element) of the copy using an XPath expression
- `b:mode` - specifies where the data is inserted **relative** to the receiving element. The attribute can have the values: `asfirstchild`, `aslastchild`, `before`, , `after`, `replace`, `replacechildren` (the default if `b:mode` is not specified).

The following is an example of how to move elements:

Example 3.7 - Moving Elements

```
<h3>What are your favorite films?</h3>
Click the films to change the order...
<ol>
    <li b:action="movedown" b:target=".">Seven Samurai</li>
    <li b:action="movedown" b:target=".">Life is Beautiful</li>
    <li b:action="movedown" b:target=".">2001: A Space Odyssey</li>
</ol>
```

3.4 Loading

A Backbase web application uses the *Single Page Interface* (SPI) model; a data-driven request-response model where each modification of the user interface results in a *data request* instead of a *page request*. Dynamically loading data fragments is therefore central to the SPI model. When you load data, you manipulate the BXML Tree by adding nodes from the server to dynamically access and update the content, structure and style of documents. Loading data allows you to organize a web application into smaller data files that are only loaded when needed.

There are several ways to load data using BXML:

- `s:include` - allows you to organize a web application into smaller data files and reuse these files more flexibly. Included files are loaded when parsed by the BPC.
- `load` command - is used to load data from an external file when needed, and provides complete control over what to load, when to load it, and where in the BXML Tree the loaded data is added.
- `b:buffer` - is used to defer loading until the buffer element is selected, the b:buffer tag only works as a child of a deck control, for example: `<b:buffer b:url="data/file.xml" />`
- `s:execute` - defining the `s:execute` tag as the root tag of your loaded file instructs

©2004-2006 Backbase B.V., All Rights Reserved.

the BPC to see the loaded file as an instruction set instead of a static content file, giving you complete control over what needs to happen when this file is loaded. For more information, see *Response Files [p. 47]*

You can use the `s:loading` control to give feedback to the users about any load action taking place. For more information, see *Loading [p. 69]*

3.4.1 Including Files

A key technique when developing an SPI application is to separate functionality into distinct modules, defined in XML files, and to include these files into your application using the `s:include` tag. Include files are well-formed XML files that contain BXML and HTML. Include files contain, for example, data, behaviors, and custom controls (styling and behavior), or they can contain a small module such as a shopping cart. Include files can also be very large and themselves contain multiple nested include files.

Using the `s:include` tag is the most common way to retrieve and place application data and resources. When the BPC finds an `s:include` tag, it loads and parses the contents before rendering the rest of the page. You include a file using the following syntax:

Example 3.8 - Including Files

```
<s:include b:url="your/data/file.xml" />
```

The Starter Kits, located in the default installation directory `C:\Backbase\3_1_4\starterkits\`, demonstrate the use of `s:include`. For example, the Forms starter kits `starterkits\forms\forms.html` startup file has the following includes:

Example 3.9 - Including Files

```
<s:include b:url="../../controls/backbase/default.xml" />
<s:include b:url="../../controls/backbase/b-button/b-button.xml" />
<s:include b:url="../../controls/backbase/b-window/b-window.xml" />
<s:include b:url="forms.xml" />
```

The first include is to `default.xml` containing miscellaneous application behavior and styling (for `b:panel` and `s:loading`). The second and third include the custom control files that define behavior and styling for the button and window controls, and the last includes is to the behavior and events used in the forms starter kit. These files are loaded, parsed and rendered when the application starts up.

3.4.2 Load Command

The `load` command allows you to load data from an external file and place the content of that file in a specified location. The `load` command has the following attributes:

- `b:url` - contains the URL targeting the data to be included.
- `b:destination` - contains an XPath expression targeting the destination node where the file is loaded. If the expression targets multiple nodes, the first element in the node list is used.
- `b:mode` - specifies where the loaded content is placed, relative to the destination. By default, the value is set to `replacechildren`.

The Windows Starter Kit, located in the default installation directory `\Backbase\3_1_4\starterkits\windows`, demonstrate the use of `load` command. The `starterkits\windows\windows.html` startup file has the following `load` commands defined within link tags. When you click a link, the file specified in the `b:url` attribute is loaded into the page as the last child of the element with the id `windowmanager`:

Example 3.10 - Load Command

```
<a b:action="load" b:url="window-01.xml" b:test="not(id('window:news'))"
b:destination="id('windowmanager')" b:mode="aslastchild">News</a>
<br />
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
<a b:action="load" b:url="window-02.xml" b:test="not(id('window:finance'))"
b:destination="id('windowmanager')" b:mode="aslastchild">Finance</a>
<br />
<a b:action="load" b:url="window-03.xml" b:test="not(id('window:weather'))"
b:destination="id('windowmanager')" b:mode="aslastchild">Weather</a>
<br />
<a b:action="load" b:url="window-04.xml" b:test="not(id('window:movie'))"
b:destination="id('windowmanager')" b:mode="aslastchild">Movies</a>
<br />
```

3.4.3 Buffered Loading

Buffered loading postpones the loading of a data resource until the state of an element is selected. Buffered loading is useful because you can defer loading, therefore reducing the initial loading time. You can use buffered loading for the deck widget using the `b:buffer` tag. For example:

Example 3.11 - Buffferd Loading

```
<b:deck id="buffer_spideck" class="b-deck">
    <b:buffer b:url="data/seven.xml" />
    <b:buffer b:url="data/2001.xml" />
    <b:buffer b:url="data/bella.xml" />
</b:deck>
<b:button b:action="previous" b:target="id('buffer_spideck')">Previous</b:button>
<b:button b:action="next" b:target="id('buffer_spideck')">Next</b:button>
```

When a deck contains buffers, it loads the contents of the URL denoted by the `b:url` attribute when it is selected for the first time and the contents of the file are cached in memory. For more information on the deck widget, see *Decks [p. 56]*.

Buffered loading cannot load files that have the `s:execute` tag as the document root.

3.4.4 HTML Entities

If you set the `b:htmlentities="true"` attribute on the `body` tag, any doctype reference in a received XML document is resolved. If there is no doctype declared in the received XML file, a standard doctype is added that references a local file (the `entities.xml` file in the root of your Backbase installation directory). This setting has a small performance penalty (it results in an extra request going to the server), therefore you should only use it when you use non-standard XML entities (the default is false).

3.5 Submitting Forms

In Backbase, you can submit forms using the standard HTML form submission, or using Backbase-enabled form submission. In both cases, standard HTTP is used for communication between the BPC and the web server using the standard HTML `action` and `method` attributes to indicate respectively the name of the file to send the form content to, and the submit method (`post` or `get`) to use.

The advantage of using Backbase form submission, rather that standard HTML form submission, is that the form submission is done using an Ajax request. You can therefore insert the result of the form submission into the current page, rather than with HTML form submission that results in a full page refresh.

This section contains the following subjects describing form submission:

- *Standard HTML Form Submission [p. 46]*
- *Backbase Form Submission [p. 46]*
- *Defining Response Files [p. 47]*
- *Using the submit Command [p. 48]*
- *Sending XML to the Server [p. 49]*

©2004-2006 Backbase B.V., All Rights Reserved.

3.5.1 Standard HTML Form Submission

In Backbase, you can can use standard HTML form submission where the submit causes the browser to go to another page when the form data is passed as query parameters or POST data, depending on the form method. The form looks and works exactly as a normal HTML form, for example:

Example 3.12 - Standard HTML Form Submission

```
<form id="main" action="response.php" method="post">
    <table>
        <tr>
            <td>Name</td>
            <td>
                <input name="name" type="text" size="40" />
            </td>
        </tr>
        <tr>
            <td>Data</td>
            <td>
                <textarea name="data" />
            </td>
        </tr>
    </table>
    <input type="submit" value="Send" />
</form>
```

For the submit button, you can use either `<input type="submit" value="Send"/>`, or `<button type="submit">Send</button>`, they both work the same.

The default type of an input element is text, and the default type of a button is usually submit, although this can vary from browser to browser. It is therefore recommended to always specify the type explicitly if you are using the standard HTML elements.

3.5.2 Backbase Form Submission

To define Backbase-enabled form submission, you must define the following attributes on the `form` element:

- `b:destination` - an XPath statement that targets an element in the BXML tree to specify at which point in your XML document the result is inserted. The destination is often the form itself, but can be any element in the document.
- `b:mode` - controls how the result of the request is inserted, relative to the given `b:destination`. Possible values are; `after`, `asfirstchild`, `aslastchild`, `before`, `replace` (the default), and `replacechildren`.

If you do not specify a `b:destination` on your `form`, a standard HTML form is submitted and **NOT** a Backbase form.

Backbase also has a `b:button` control that you can use to submit the form that has default styling applied, standardizing the button's appearance across browsers. The `b:button` requires that you use slightly different attributes to the standard input or button type:

- `b:action="submit"` - the action that triggers the submittal process
- `b:target` – requires an XPath expression to identify the form that is submitted (usually the parent form)

The following is an example of Backbase form submission:

Example 3.13 - Backbase Form Submission

```
<form id="main" b:destination="." b:mode="replace" action="response.php" method="post">
    <table>
        <tr>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
            <td>Name</td>
            <td>
                <input name="name" type="text" size="40" />
            </td>
        </tr>
        <tr>
            <td>Data</td>
            <td>
                <textarea name="data" />
            </td>
        </tr>
    </table>
    <b:button b:action="submit" b:target="id('main')">Send</b:button>
</form>
```

Where:

- `b:destination="."` defines that the response is inserted in the current context node (the form)
- `b:mode="replace"` indicates the response replaces the form element in the BXML tree
- `b:button b:action="submit" b:target="id('main')"` submits the element by targetting the element whose `id` is `main` (the form)
  You can also target the form using a relative XPath expression, for example `../../form`, or by using `"ancestor::form[1]"`, which ensures that you get the parent form no matter how many elements are in-between.

You can actually use `b:action="submit" b:target="id('main')"` on any element. However, you must not use it on an HTML `input type="submit"` element otherwise the form is sent twice. The same applies to the HTML `button` element that also frequently has a default submit action.

3.5.3 Defining Response Files

A response file is a BPC file that is sent as the response when a form is submitted. The XML standard requires that the document contains only one root node which defines the namespaces it uses. The file does not need contain an XML declaration, or a DOCTYPE declaration.

The following is an example of a response file that is used when a form is submitted. When you submit the form, the result of the request replaces the element, often the `form` element, that is targeted using the `b:destination` attribute. The result of a form submission must be XHTML or BXML. For example:

Example 3.14 - Response File

```
<div id="formresponse" xmlns="http://www.w3.org/1999/xhtml"
xmlns:b="http://www.backbase.com/b" xmlns:s="http://www.backbase.com/s">
<p>Thank you for entering your details...</p>
</div>
```

The response file must include namespace declarations for the XHTML and BXML elements it contains. It is good practice to declare the XHTML and BXML `s:` and `b:` namespaces in the root element, even if the file does not currently use any of these elements, in case these elements are added later. (Note that in Firefox, the BPC adds BXML namespaces to the XML even if they are not declared. However, in Internet Explorer the namespaces are not added, therefore not declaring the namespaces causes the load to fail.)

The server can respond with data, as with the example above, or with an instruction set by defining the `s:execute` tag as the root tag of your reponse file. `s:execute` instructs the BPC to see the loaded file as an instruction set instead of a static data file, giving you complete control over what needs to happen when the file is loaded.

For example, when a form is submitted, you often want to leave the interface as it is and

©2004-2006 Backbase B.V., All Rights Reserved.

inform the user, using for example a modal dialog, whether the form was successfully submitted. By defining a modal dialog in the response that is sent, information is not lost and the user can correct errors in the form. The following provides an example of defining a response file using `s:execute` (the response is rendered as the last child in the `body` element):

Example 3.15 - Response File Using s:execute

```
<s:execute xmlns:s="http://www.backbase.com/s" xmlns:b="http://www.backbase.com/b"
xmlns="http://www.w3.org/1999/xhtml">
<s:render b:destination="/*" b:mode="aslastchild">
    <b:modal id="response-modal">
        <b:modalhead>Form response</b:modalhead>
        <b:modalbody>
            <div>We have received your form. Thank you for your order.</div>
            <br />
            <div>Currently when you close this modal window, the form stays visible. In a
real                world application you would                use a command to overwrite
the form when the form was succesfully submitted.</div>
            <br />
            <br />
            <br />
            <center>
                <b:button b:action="trigger" b:event="close"
b:target="ancestor::b:modal[1]">Close</b:button>
            </center>
        </b:modalbody>
    </b:modal>
</s:render>
<s:task b:action="select" b:target="id('response-modal')" />
</s:execute>
```

The `s:execute` tag contains only execution tags and executes its children immediately after being parsed. When you define `s:execute` as the root of a response file loaded by a form submission (or a `load` command), the data is **NOT** inserted at the specified destination but is executed immediately. You must therefore use the `s:render` tag to specify where the data needs to be placed using the `b:destination` and `b:mode` attributes.

3.5.4 Using the Submit Command

When a form is submitted, a `submit` event is triggered on the form. This allows you to "intercept" the form submission and perform, for example, field validation before submitting the form. To perform field validation, or other pre-submittal actions, you need to define an event handler for the `submit` event. You must add custom code in the `submit` event handler that validates the form and determines whether the form is valid and can be sent to the server. The event handler must also include a `send` command to actually submit the form after the validation has taken place.

The following example is from the Forms Starter Kit located in the default Backbase installation directory `C:\Backbase\3_1_4\starterkits\forms\form.html`. The form `b:behavior` attribute indicates that the form uses a behavior called `form`, defined in the `forms.xml` file, that defines an event handler for the `submit` event:

Example 3.16 - Form Submit

```
<form action="response.xml" b:destination="." method="post" b:behavior="form"></form>
```

The `submit` event handler in the `forms.xml` includes a `send` command that is triggered when the result of the `b:test` is successful (form validation is implemented in the form):

Example 3.17 - Submit Event Handler

```
<s:behavior b:name="form">
    <s:event b:on="construct"></s:event>
    <s:event b:on="submit">
        <s:choose>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
            <s:when b:test=".//b:step">
                <s:task b:action="send" />
            </s:when>
            <s:otherwise>
                <s:task b:action="trigger" b:target=".//input" b:event="validate" />
                <s:choose>
                    <s:when b:test="not(.//input[@b:valid][@b:valid != 'true'])">
                        <s:task b:action="alert" b:value="valid" />
                        <s:task b:action="send" />
                    </s:when>
                    <s:otherwise>
                        <s:task b:action="alert" b:value="invalid" />
                    </s:otherwise>
                </s:choose>
            </s:otherwise>
        </s:choose>
    </s:event>
</s:behavior>
```

3.5.5 Sending XML to the Server

The BPC can also send XML to the server. To send XML to the server, target a BXML element, or set of elements, and issue the `send` command: the element and all descending elements are converted to XML and sent using the `HTTP POST` command with the `text/xml` encoding. An XML-enabled server can then load the XML directly into an XML document (the location of the server file is defined using the `b:url` attribute).

The action associated with the `send` command depends on which elements are targeted. The `send` command uses the `b:source` attribute to specify which element, or elements, need to be sent. If a `form` element is targeted, the form is submitted in the default form manner. If any other element is targeted, the element is sent as XML.

To allow you to target multiple source nodes (containing multiple root nodes), the generated XML is wrapped in a `b:backbase` tag to create valid XML. For example:

Example 3.18 - Sending XML to the Server

```
<table>
    <s:event b:on="command">
        <s:task b:action="send" b:source="tbody/tr" b:url="receiveXML.php" />
    </s:event>
    <thead>
        <tr>
            <th>Movie</th>
            <th>Genre</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>The Seven Samurai</td>
            <td>Drama</td>
        </tr>
        <tr>
            <td>The Godfather</td>
            <td>Crime</td>
        </tr>
    </tbody>
</table>
```

Is received on the server as:

Example 3.19 - XML that is sent to the server

```
<b:backbase>
    <tr>
        <td>The Seven Samurai</td>
        <td>Drama</td>
    </tr>
    <tr>
        <td>The Godfather</td>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <td>Crime</td>
    </tr>
</b:backbase>
```

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 4 - Asynchronous Execution

4.1 Asynchronous Execution

Events add interactivity to your SPI: when the user does something an event takes place (although there are also some events that are not directly caused by the user, for example the `construct` event fires when a page has been constructed). In Backbase, you can attach an event to any element in the UI. You must then create an event handler to determine what takes place when the event is triggered. An event handler is a container for executing commands. It handles the event by executing a command, or series of commands.

A Backbase application uses the *Asynchronous Execution* system: each event that is triggered is executed *asynchronously* in a separate thread. By default, therefore, multiple command threads can be active at the same time.

It is important to realize, however, that the commands defined within each event handler are executed by default *synchronously*. Within each thread therefore, the commands are, by default, executed sequentially (one by one).

All commands also trigger events of the same name. For example, an alert command triggers an alert event. For more information, see *Defining Event Handlers [p. 87]*.

You can overrule the default asynchronous/sychronous command execution using the following BXML tags:

- `s:lock` tag - you can lock the processes thread on a specfied target element, in which case each event triggered is queued and executed synchronously. If, in the meantime, events are triggered on elements that do not have a lock applied, those events are executed asynchronously (the lock only applies to a target element.
- `b:async` attribute - within a thread, you can specify that a command execute asynchronously by setting the attribute `b:async="true"`
- `s:parallel` - specify that `s:fxstyle` commands run in parallel
- `s:sequential` - specify that `s:fxstyle` commands run sequentially

In the following example, the behavior defines an event handler for the `select`, and one for the `deselect`. Each event triggers a new command, executed *asynchronously* in its own thread. Within the event handler, the commands are executed *synchronously*:

Example 4.1 - Asynchronous Execution

```
<s:behavior b:name="diveffect">
    <s:event b:on="select">
        <s:fxstyle b:background-color="#ffff00" b:time="5000" b:async="true" />
        <s:fxstyle b:width="~50px" b:height="~50px" />
        <s:fxstyle b:left="~50px" />
        <s:fxstyle b:left="~50px" />
    </s:event>
    <s:event b:on="deselect">
        <s:fxstyle b:width="~-50px" b:height="~-50px" />
        <s:fxstyle b:left="~-50px" />
        <s:fxstyle b:left="~-50px" />
    </s:event>
</s:behavior>
<p>
    <a b:action="select" b:target="id('changingdiv')"> Click         Here to Start</a>
</p>
<div>
    <div style="position: relative; background-color: #0000ff; height: 50px; width: 50px;"
b:behavior="diveffect" id="changingdiv"></div>
    <p>
        <a b:action="deselect" b:target="id('changingdiv')">             Click Here to
End</a>
    </p>
</div>
```

When you click the link "Click Here To Start", the `select` command is triggered on the

©2004-2006 Backbase B.V., All Rights Reserved.

div element id="changingdiv", which in turn triggers the event handler defined in the behavior. The select event handler asynchronously changes color (the s:fxstyle has the b:async="true" attribute), and the other fxstyles to change size and position each execute synchronously. The effect is that the element's background color is changing even while it is being resized and moved.
When you click the link "Click Here To End", the deselect command is triggered on the div element id="changingdiv", which in turn triggers the deselect event handler defined in the behavior. The fxstyles are executed synchronously; one transition effect needs to end before the next is started. (If the asynchronous transition effect is still being executed, even in its deselected state, the element will still keep changing color.)

For more information on effects, see *Controlling Transitions [p. 106]*

Entanglement: When you declare an action to run asynchronously, the result of the new thread are ignored by the BPC. Therefore, you must carefully consider whether the response, coming from the s:fxstyle, s:task, s:parallel, s:sequential and s:event tags extended with the b:async attribute, can safely be ignored by the BPC. For example, you can issue a set command asynchronously as long as you are aware that other threads in the same process cannot be targeting elements by the response or the result from this command - the new attribute value.

4.1.1 Sequential And Parallel Processing

You can specify whether the BPC runs the s:fxstyle commands in parallel, or sequentially (one at a time).

Implicitly, s:fxstyle commands are executed sequentially when you create a separate s:fxstyle tag for each style attribute you want to animate, that is you place them one after the other, the transitions run one by one. For example:

Example 4.2 - Sequential Processing

```
<s:event b:on="select">
    <s:fxstyle b:font-size="20px" />
    <s:fxstyle b:color="#33FF33" />
</s:event>
```

You can also implicitly execute s:fxstyle commands in parallel by defining multiple styles in a single s:fxstyle tag, in which case all the style transitions are animated in parallel. For example:

Example 4.3 - Multiple Inline Style Effects

```
<s:event b:on="select">
    <s:fxstyle b:font-size="20px" b:color="#33FF33" />
</s:event>
```

You can explicitly control whether s:fxstyle (and s:task tags) are executed in parallel or sequentially by grouping them between s:sequential or s:parallel tags. In the following example, the first transitions for changing the font-size and color change in parallel, and when completed the font-size and the color return to their original values sequentially:

Example 4.4 - Sequential and Parallel Processing

```
<s:event b:on="command">
    <s:parallel>
        <s:fxstyle b:font-size="20px" />
        <s:fxstyle b:color="#ff0000" />
    </s:parallel>
    <s:sequential>
        <s:fxstyle b:font-size="12px" />
        <s:fxstyle b:color="#0000ff" />
    </s:sequential>
</s:event>
```

Note: Since sequential is the default behavior, you can omit the s:seqential tags from

©2004-2006 Backbase B.V., All Rights Reserved.

the s:fxstyle tags.

4.2 Locking For Explicit Synchronization

The s:lock tag is used for thread locking to prevent two events executing simultaneously, forcing the events to execute sequentially. You can lock any element in the document tree using the b:target attribute. The example below demonstrates the principles behind locking, and the Asynchronous Command Execution for event execution.

The following list describes the elements defined in the example below:

- s:behavior - the lock-move behavior contains two event handlers, one to handle the moverright event, and the other on the moveleft event. The moveright event handler has a lock that uses relative XPath targeting to target an element
- XHTML link elements - the first a tag contains a b:action trigger command that is used to trigger the moveright event. The b:target="~+table[1]/tbody/tr" is an XPath expression that targets, starting from the context node (the a element), the first sibling element that a table element, and selects all the table row elements in the table body.
- XHTML table elements - the first table contains four rows, and each row uses the lock-move behavior. The second table is at start up, an empty table that is the destination target of the moveright event.

Example 4.5 - Locking

```
<s:behavior b:name="lock-move">
    <s:event b:on="moveright">
        <s:lock b:target="..">
            <s:task b:action="move" b:source="." b:destination="../../~+table[1]/tbody"
b:mode="aslastchild" />
            <s:fxstyle b:cancel="false" b:time="500" />
        </s:lock>
    </s:event>
    <s:event b:on="moveleft">
        <s:task b:action="move" b:source="." b:destination="../../~-table[1]/tbody"
b:mode="aslastchild" />
        <s:fxstyle b:cancel="false" b:time="500" />
    </s:event>
</s:behavior>
<a b:action="trigger" b:event="moveright" b:target="~+table[1]/tbody/tr" style="position:
absolute;">Move rows to the right (locked)</a>
<a b:action="trigger" b:event="moveleft" b:target="~+table[2]/tbody/tr" style="position:
absolute;left:300px;">Move rows to the left (not locked)</a>
<table class="filmTable" style="position: absolute; top:25px;">
    <thead>
        <tr>
            <th style="width: 150px;">Movie</th>
            <th style="width: 60px;">Genre</th>
        </tr>
    </thead>
    <tbody>
        <tr b:behavior="lock-move">
            <td>The Seven Samurai</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>City of God</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>Donnie Darko</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>The Godfather</td>
            <td>Crime</td>
        </tr>
    </tbody>
</table>
<table class="filmTable" style="position: absolute;left: 300px; top:25px;">
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    <thead>
        <tr>
            <th style="width: 150px;">Movie</th>
            <th style="width: 60px;">Genre</th>
        </tr>
    </thead>
    <tbody />
</table>
```

The `moverright` and `moveleft` events are executed as follows:

1. Click the "Move rows to the right (locked)" link to trigger the `moverright` event. The event targets the table rows in the first table (the target element is therefore contains a node-set).
2. Each row in the table triggers the `moveright` event handler, defined in the behavior `lock-move`. Therefore, for each row, a separate thread is created to handle the event. However, because the event handler locks the `tbody` element (using XPath targeting), each thread runs synchronously-each thread must complete before the next thread is started.
3. The `moveright` event handler takes the first table row and inserts it at the location designated by the `b:destination` attribute, in this case, `../../~+table[1]/tbody` means it navigates from the context node (the `tr` element in the first table) up two levels to the `table` element, finds the first sibling table, and inserts the `tr` element as the last child in the `tbody`.
4. When the first table row is moved, it executes the event handler on the next `tr` element, and so on. An `fxstyle` is defined on the event handler, defining a time of 500 milliseconds, that delays the handling of the event sufficiently for you to see how each each event handler is triggered individually, and is completed before the next event is triggered, creating an animation effect.
5. When you trigger the `moveleft` event handler, exactly the same happens as in the `moveright` event handler: for each row, a separate thread is created to handle the event. However, because no lock is specified, the execution of each event is asynchronous - all the events are processed at the same time creating an effect that the rows are moved in one block, when in fact they are all moved individually.

4.3 Aborting

You can set the `b:cancel` attribute value to `false` to prevent the user from being able to stop an `s:fxstyle` effect. When you set `b:cancel="true"`, the default, the user can stop the effect. The effect is stopped at its current location.

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 5 - Creating Application Layout

5.1 Defining Screen Partitioning

Screen Partitioning determines the basic layout of the user interface. You define basic screen partitioning using normal XHTML tags such as `<table>`, `<div>` and `<span>`.

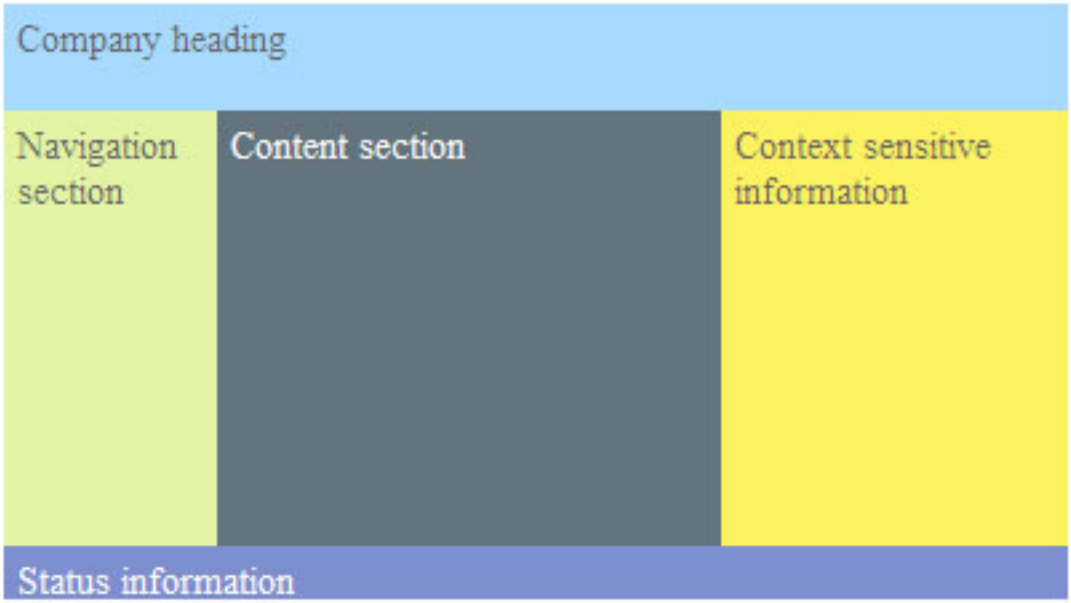
You can define advanced screen partitioning using the BXML tags `b:panel` and `b:deck`, and using the Backbase layout client controls `b:window`, `b:spacer`, `b:modal`, `b:box`, and `b:tile`.

5.1.1 Panelsets and Panels

Panelsets allows you to divide a screen up into rows or columns (using a `b:row` or a `b:cols` attribute. A panelset is a container for panels; a row or column in a panelset is represented by a panel. You define the content in the panel

In the following example, a panelset is defined that contains three rows. The first (Company Heading) and third (Status information) rows, contain panels. The second row is a nested panelset that defines three columns, and each one of the columns (Navigation section, Content section, and Context sensitive information section) is defined using a panel.

Figure 5.1 - Backbase Panelsets



The rows and columns have either a fixed width, or they can have a relative width given as a percentage of the currently available width of the panelset parent. You can specify the following units of measurement for rows and columns in panelsets:

- `%` (percentage) units are relative to the parent element.
- `em` and `ex` units are relative to the surrounding text.
- `px` (pixels) units are relative to the screen resolution.
- `pt` (point) and `pc` (pica) are print units.
- `in`, `cm` and `mm` (inches, centimeters and millimeters) are length units.

Alternatively, a panel can fill up the remaining available space using the wildcard asterisk (*) sign. You must specify all values before or after a wildcard (asterisk) using the same unit of measurement. For example, `b:cols="50px 100px *"` is valid but NOT `cols="50% 100px *"`

©2004-2006 Backbase B.V., All Rights Reserved.

You can only define rows or columns, but not both, within a panelset (instead, define nested panelsets).

In the following example, the root panelset has three rows; the top row is 40 pixels, the botton row is 20 pixels, and the middle row takes up the remaining space. Each row in the panelset is defined using a panel. The middle row contains a nested panelset that divides the row into three columns; the first column is 80 pixels, the last column is 130 pixels, and the middle row takes up the remaining space.

Example 5.1 - Defining Panelsets

```
<b:panelset b:rows="40px * 20px" style="color: white;" b:minwidth="300px"
b:minheight="200px">
    <b:panel style="background-color: #A8DAFF; color: #666; padding: 5px 0 0 5px">Company
heading</b:panel>
    <b:panelset b:cols="80px * 130px">
        <b:panel style="background-color: #E3F5A5; color: #666; padding: 5px 0 0
5px">Navigation section</b:panel>
        <b:panel style="background-color: #64747F; color: white; padding: 5px 0 0
5px">Content section</b:panel>
        <b:panel style="background-color: #FDF260; color: #666; padding: 5px 0 0 5px">Context
sensitive information</b:panel>
    </b:panelset>
    <b:panel style="background-color: #7D8FCE; color: white; padding: 5px 0 0 5px">Status
information</b:panel>
</b:panelset>
```

Internet Explorer performs slower if ancestor elements of a panelset have the style attribute `height: 100%`. To resolve this, set the parent elements, that have 100% height or 100% width, to `overflow: auto` or `overflow: hidden`

5.1.2 Decks

A deck is an element that displays only one of its children at a time, with the common purpose of creating tabbed or wizard-like interfaces. To implement a simple tab-like deck:

- Create the sections of the deck (usually as `div` tags or separate files). These sections are child elements of the deck. Of these sections the default one is the section that appears first when the deck is initially rendered.
- Create links or buttons to flip through the deck; and define attributes such as color, size, positioning.
- Add commands to these buttons to address the navigation functionality of the deck. This can include commands for next, previous, last page, first page, selection, deselection, and so on.

Example 5.2 - Defining a Deck

```
<div>
    <b:deck class="simpleDeck">
        <div>1 This is a deck ...</div>
        <div>2... a widget to create ...</div>
        <div>3... multi-paged documents.</div>
    </b:deck>
    <a b:action="previous" b:target="../b:deck">&lt; Previous</a>
    <a b:action="next" b:target="../b:deck">Next &gt;</a>
</div>
```

To load the contents of a deck, where each section is stored in a separate file, you can use the `s:include` tag, although this has the disadvantage that the files are loaded immediately. To delay the loading, you can instead use the `b:buffer` tag. For more information on buffered loading, see *Buffered Loading [p. 45]*

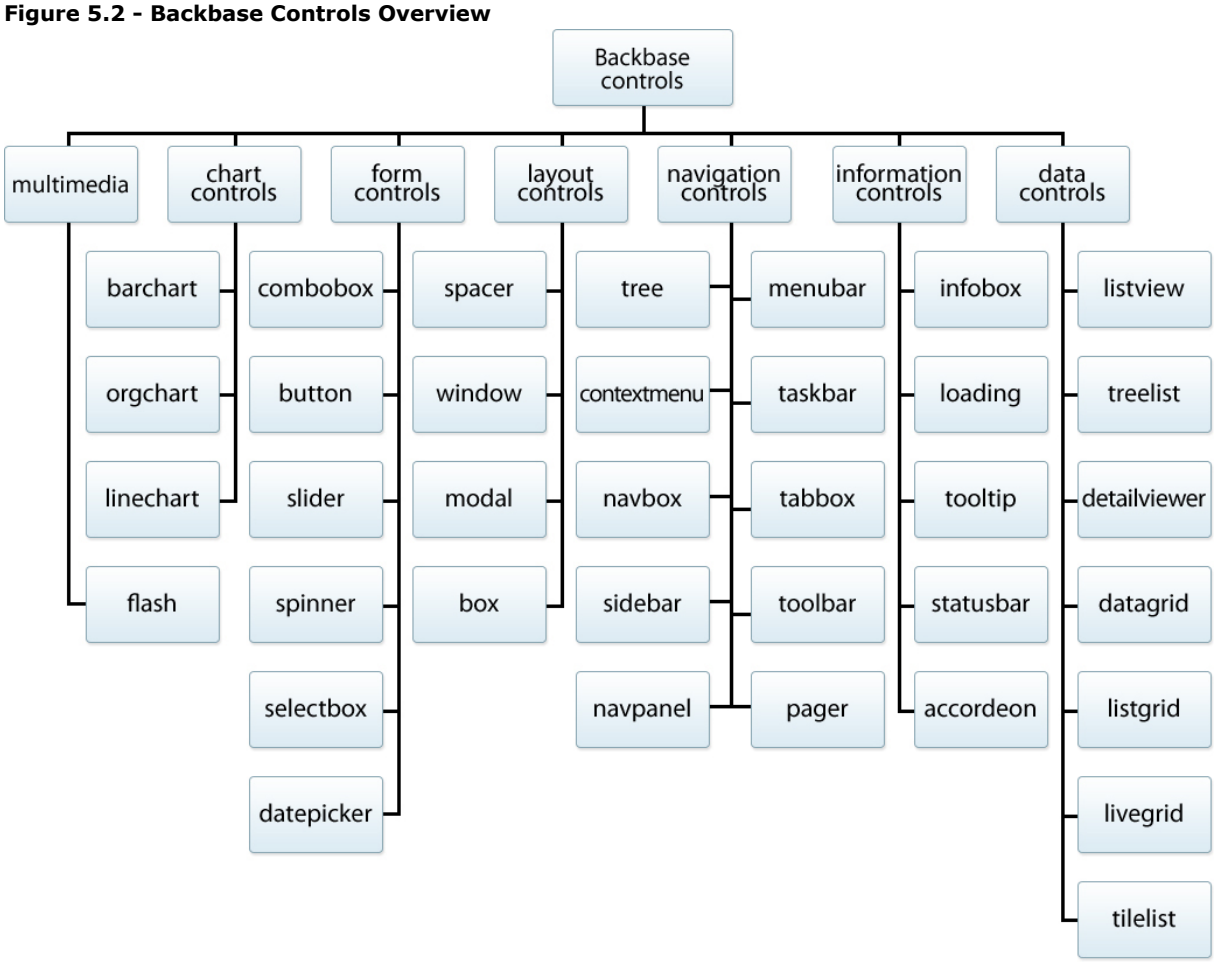
Deck child elements can follow the state changes of yet another deck and select or deselect themselves when they are selected or deselected using the `b:followstate` relationship. This allows you to declare complex relations between different user interface controls. *b:followstate [p. 103]*

©2004-2006 Backbase B.V., All Rights Reserved.

5.2 Backbase Controls Overview

Backbase Controls are the Backbase out-of-the-box, reusable user interface controls that encapsulate common patterns and behaviors. The following illustration provides an overview and categorization of the controls, which are described in the following sections:

Figure 5.2 - Backbase Controls Overview



You need to set the `b:controlpath` attribute in the `body` tag to specify the controls you want to use. For more information, see *Setting the Control Path [p. 84]*.

5.3 Defining Layout Controls

©2004-2006 Backbase B.V., All Rights Reserved.

Use the layout controls to determine the basic layout of the user interface. You can define advanced screen partitioning using the BXML client controls: `b:box`, `b:modal`, `b:spacer`, and `b:window`.

5.3.1 Box

A box control is a simple component that can only contain inline elements. You can give it a height or width; if no size is specified, its size is derived from its contents. You can style the innercontent of the box using the `b:style` attribute:

Figure 5.3 - Backbase Box

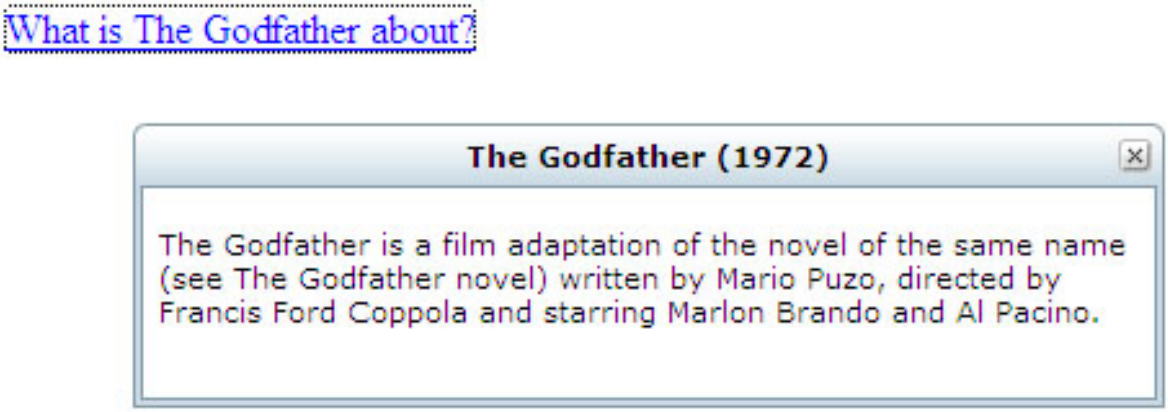
Example 5.3 - Creating a Box Control

```
<b:box style="width: 250px;" b:style="background-color: #CCFF00">      "Scarface developed a cult following among younger audiences."</b:box>
```

5.3.2 Modal

The modal control is a pop-up window. When opened, the user cannot interact with the rest of the document until the modal window is closed. You can open the modal dialog by triggering (using the `trigger` command) the `open` event, and close it by triggering the `close` event. The Modal also has a default button in the top right-hand corner for closing the window.

Figure 5.4 - Backbase Modal


What is The Godfather about?
The Godfather (1972)
The Godfather is a film adaptation of the novel of the same name (see The Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon Brando and Al Pacino.

Example 5.4 - Creating a Modal Control

```
<div>
    <p>
        <a b:action="trigger" b:event="open" b:target="id('modal')">What is The Godfather about?</a>
    </p>
    <b:modal id="modal">
        <b:modalhead>The Godfather (1972)</b:modalhead>
        <b:modalbody>
            <p>The Godfather is a film adaptation of the novel of the same name (see The Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon Brando and Al Pacino.</p>
            <a b:action="trigger" b:event="close" b:target="id('modal')">Close the Modal</a>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        </b:modalbody>
    </b:modal>
</div>
```

5.3.3 Spacer

The `b:spacer` control is an invisible element that is used to separate elements. It has a `b:width` and `b:height` attribute that specifies a percentage or pixel value. Spacers allow you to separate items to pixel precision.

Example 5.5 - Creating a Spacer Control

```
<p>The following text has spacers inside.
    <br />
    Hello
    <b:spacer b:width="200px" b:height="150px" />
there!
    <b:spacer b:width="3em" />
How
    <b:spacer b:width="10px" />
are
    <b:spacer b:width="2px" />
you?
</p>
```

5.3.4 Window

A `b:window` tag creates a window that you can move, similar to windows in desktop applications. The child tags `b:windowhead` and `b:windowbody` define the head and body of the window.

Figure 5.5 - Backbase Windows

Open window 1 | Open window 2
Life Is Beautiful (La Vita è Bell
A 1997 Italian language film which tells
the story of an Italian Jew
2001: A Space Odyssey
An influential science fiction film
directed by Stanley Kubrick, notable
for combining episodes contrasting high
levels of scientific and technical realism

In the following example, two windows are created, one of which is, by default, opened on startup. The windows contain buttons in the top right-corner, defined in the `b:windowbuttons` attribute, for closing, minimizing, and maximizing the window. Link elements are defined that allow you to open the windows: the `trigger` command triggers an `b:event="open"` to open the window specified using an XPath expression (`b:target="id('win1')"`):

©2004-2006 Backbase B.V., All Rights Reserved.

Example 5.6 - Creating a Window Control

```
<div style="position: relative; height: 100%; overflow:visible; border : 1px solid red;">
    <a b:action="trigger" b:event="open" b:target="id('win3')">Open window 1</a>
    |
    <a b:action="trigger" b:event="open" b:target="id('win4')">Open window 2</a>
    <b:window id="win3" b:open="true" style="top:100px; left:20px;">
        <b:windowhead>Life Is Beautiful (La Vita è Bella)</b:windowhead>
        <b:windowbody>A 1997 Italian language film which tells the story of an Italian
Jew...</b:windowbody>
    </b:window>
    <b:window id="win4" b:open="false" style="top:150px; left:100px;">
        <b:windowhead>2001: A Space Odyssey</b:windowhead>
        <b:windowbody>An influential science fiction film directed by Stanley Kubrick,
            notable for combining episodes contrasting high levels of scientific and
            technical realism with transcendental mysticism.</b:windowbody>
    </b:window>
</div>
```

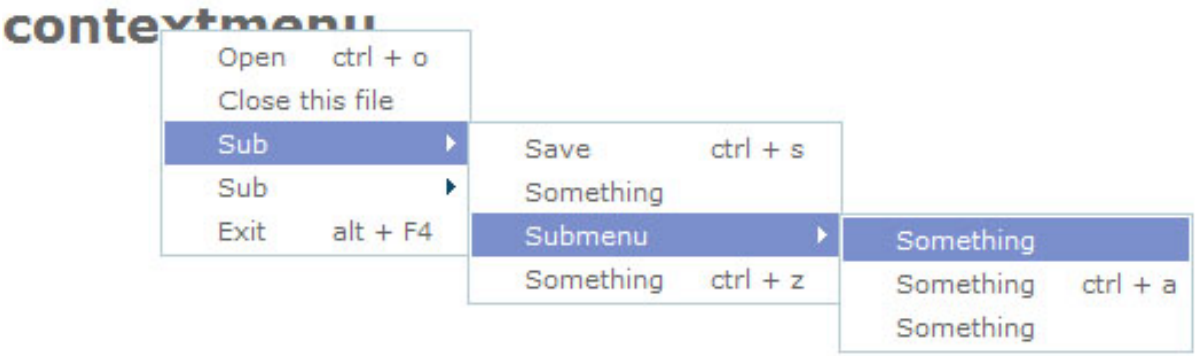
5.4 Defining Navigation Controls

Use the navigation controls to determine the basic navigational requirements layout of the user interface. You can define navigation using the BXML client controls: `b:contextmenu`, `b:menubar`, `b:navbox`, `b:navpanel`, `b:sidebar`, `b:tabbox`, `b:taskbar`, `b:toolbar`, and `b:tree`.

5.4.1 Contextmenu

The Contextmenu is a context-sensitive menu that pops-up when an event, typically a right-mouse button event, is triggered on an element. You create the menu by nesting `b:contextmenu` tags, and you define each row in the menu using the `b:contextmenurow` tag. The `b:contextmenurow` tag has a `b:label` attribute that defines the menu label, and optionally a `b:shortcut` attribute that adds a label to denote a shortcut key, for the menu (you must implement the keys using the `s:keys` tag).

Figure 5.6 - Backbase Contextmenu



In the following example, a right-mouse button (`rmbup`) event handler is defined that targets the context menu. Right-click the heading element to view the pop-up menu.

Example 5.7 - Creating a Contextmenu Control

```
<div>
    <h1>contextmenu</h1>
    <s:event b:on="rmbup" b:bubble="false">
        <s:task b:action="select" b:target="id('contextmenu')" />
        <s:task b:action="position" b:type="place" b:target="id('contextmenu')"
b:destination="." b:position="at-pointer" />
    </s:event>
    <b:contextmenu id="contextmenu">
        <b:contextmenurow b:label="Open" b:shortcut="ctrl + o" />
        <b:contextmenurow b:label="Close this file" />
```

©2004-2006 Backbase B.V., All Rights Reserved.

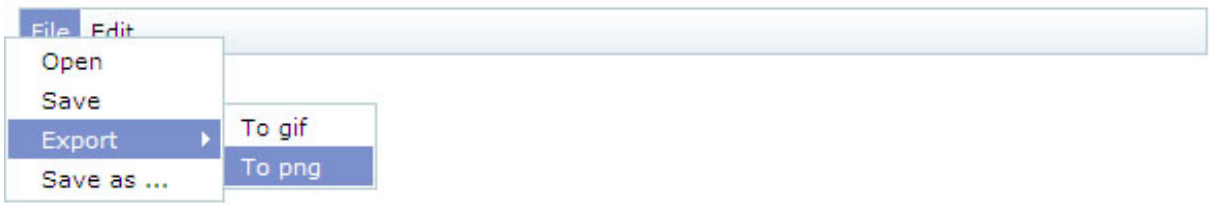
```
        <b:contextmenurow b:label="Sub">
            <b:contextmenu>
                <b:contextmenurow b:label="Save" b:shortcut="ctrl+s" />
                <b:contextmenurow b:label="Something" />
                <b:contextmenurow b:label="Submenu">
                    <b:contextmenu>
                        <b:contextmenurow b:label="Something" />
                        <b:contextmenurow b:label="Something" b:shortcut="ctrl+a" />
                        <b:contextmenurow b:label="Something" />
                    </b:contextmenu>
                </b:contextmenurow>
                <b:contextmenurow b:label="Something" b:shortcut="ctrl+z" />
            </b:contextmenu>
        </b:contextmenurow>
        <b:contextmenurow b:label="Sub">
            <b:contextmenu>
                <b:contextmenurow b:label="Save" b:shortcut="ctrl+s" />
                <b:contextmenurow b:label="Something" />
                <b:contextmenurow b:label="Submenu" b:disabled="true">
                    <b:contextmenu>
                        <b:contextmenurow b:label="Something" />
                        <b:contextmenurow b:label="Something" b:shortcut="ctrl+a" />
                        <b:contextmenurow b:label="Something" />
                    </b:contextmenu>
                </b:contextmenurow>
                <b:contextmenurow b:label="Something" b:shortcut="ctrl+z" />
            </b:contextmenu>
        </b:contextmenurow>
        <b:contextmenurow b:label="Exit" b:shortcut="alt+F4" />
    </b:contextmenu>
</div>
<div b:dragreceive="true"></div>
```

For more information about implementing keys, see *Capturing Keys [p. 93]*.

5.4.2 Menubar

The Menubar control displays a menubar as seen in many applications. It is usually presented at the top of the web application, but you can position it anywhere.

Figure 5.7 - Backbase Menubar



The following example describes the syntax to construct a menubar:

Example 5.8 - Creating a Menubar Control

```
<s:include b:url="../controls/backbase/b-menubar/b-menubar.xml" />
<div style="width: 100%; height: 100%; position: relative">
    <div style="border-color: #b8ced7; border-width: 1px 1px 0 1px; border-style: solid;
height:23px;overflow:hidden;">
        <b:menubar>
            <b:menurow b:label="File">
                <b:menupopup>
                    <b:menupane>
                        <b:menurow>
                            <b:menucell>Open</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Save</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Export</b:menucell>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
                            <b:menupopup>
                                <b:menupane>
                                    <b:menurow>
                                        <b:menucell>To gif</b:menucell>
                                    </b:menurow>
                                    <b:menurow>
                                        <b:menucell>To png</b:menucell>
                                    </b:menurow>
                                </b:menupane>
                            </b:menupopup>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Save as ...</b:menucell>
                        </b:menurow>
                    </b:menupane>
                </b:menupopup>
            </b:menurow>
            <b:menurow>
                <b:menucell>Edit</b:menucell>
                <b:menupopup>
                    <b:menupane>
                        <b:menurow>
                            <b:menucell>Copy</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Cut</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Paste</b:menucell>
                        </b:menurow>
                    </b:menupane>
                </b:menupopup>
            </b:menurow>
        </b:menubar>
    </div>
</div>
```

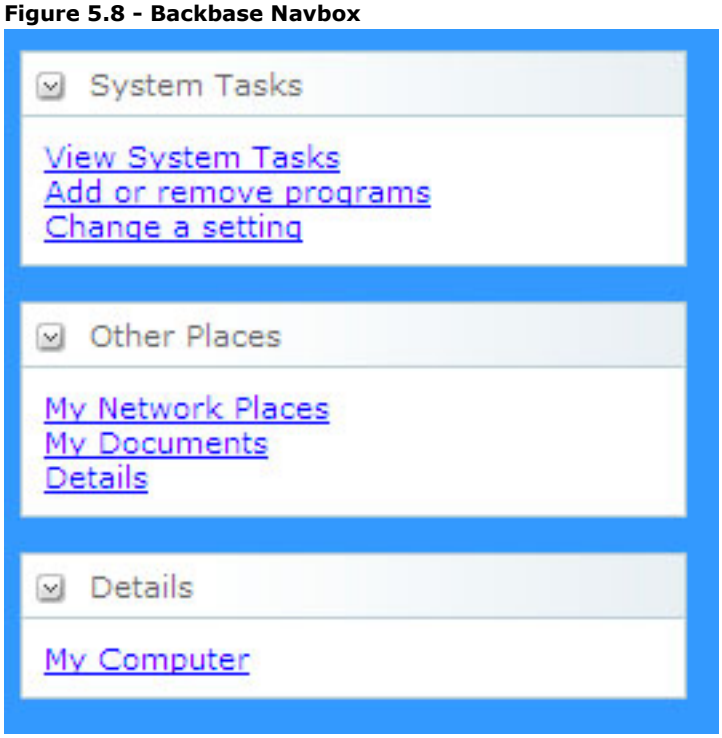
To use the menubar control, you must add an `s:include` to the path of this control, for example `s:include b:url="../controls/backbase/b-menubar/b-menubar.xml"`

5.4.3 Navbox

The Navbox control is useful for showing and hiding levels of sub-navigation and further detail. A `b:navbox` contains a single `b:navhead` and `b:navbody` element; clicking the head expands or collapses the body element in which the content is defined. The navboxes are block level, meaning that they are presented beneath one other and occupy the entire width of an area. Navboxes are designed for secondary navigational purposes; each navbox is a separate element that typically contains additional information loosely related to the others.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.8 - Backbase Navbox



Example 5.9 - Creating a Navbox Control

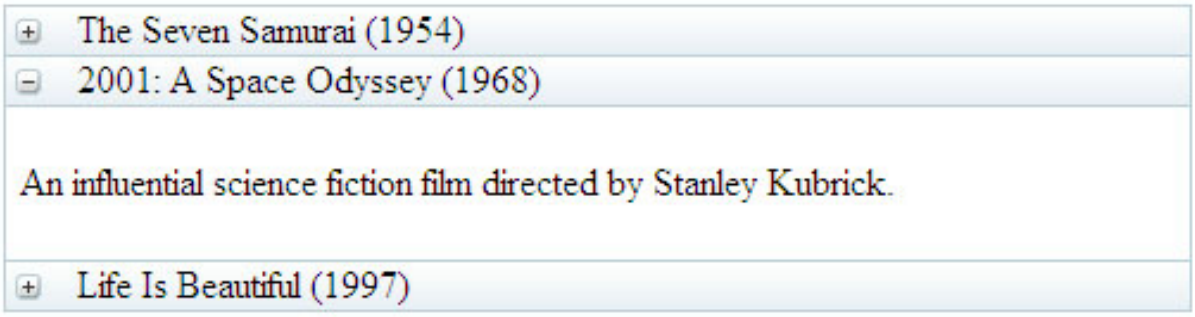
```
<div style="background-color:#3399FF; width: 280px; padding: 10px; height: 300px">
    <b:navbox style="width: 250px;">
        <b:navhead>System Tasks</b:navhead>
        <b:navbody>
            <a>View System Tasks</a>
            <br />
            <a>Add or remove programs</a>
            <br />
            <a>Change a setting</a>
        </b:navbody>
    </b:navbox>
    <br />
    <b:navbox style="width: 250px;" b:open="true">
        <b:navhead>Other Places</b:navhead>
        <b:navbody>
            <a>My Network Places</a>
            <br />
            <a>My Documents</a>
            <br />
            <a>Details</a>
        </b:navbody>
    </b:navbox>
    <br />
    <b:navbox style="width: 250px;">
        <b:navhead>Details</b:navhead>
        <b:navbody>
            <a>My Computer</a>
        </b:navbody>
    </b:navbox>
</div>
```

5.4.4 Navpanel

The Navpanel is a control that the user can expand or collapse by clicking the head and can be used for the main navigation of a web application. The navpanel assumes the full the height and width of its parent element. The `b:navpanelbody` contains the content. A `b:navpanel` typically has multiple `b:navpanelhead` and `b:navpanelbody` elements, only one of which can be selected at any time: selecting one automatically deselects the other.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.9 - Backbase Navpanel


The Seven Samurai (1954)
2001: A Space Odyssey (1968)
An influential science fiction film directed by Stanley Kubrick.
Life Is Beautiful (1997)

In the following example, the navpanel contains three navpanelheads and navpanelbodies. You can use the `b:state` attribute to define which of the heads is opened on startup.

Example 5.10 - Creating a Navpanel Control

```
<div style="height: 500px; width: 200px">
    <b:navpanel b:singular="strict">
        <b:navpanelhead>The Seven Samurai (1954)</b:navpanelhead>
        <b:navpanelbody>
            <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
        </b:navpanelbody>
        <b:navpanelhead>2001: A Space Odyssey (1968)</b:navpanelhead>
        <b:navpanelbody>
            <p>An influential science fiction film directed by Stanley Kubrick.</p>
        </b:navpanelbody>
        <b:navpanelhead b:state="selected">Life Is Beautiful (1997)</b:navpanelhead>
        <b:navpanelbody>
            <p>A Italian language film which tells the story of an Italian Jew, Guido Orefice
(played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use
that dreamy quality to survive a concentration camp with his young son Joshua (played by
Giorgio Cantarini).</p>
        </b:navpanelbody>
    </b:navpanel>
</div>
```

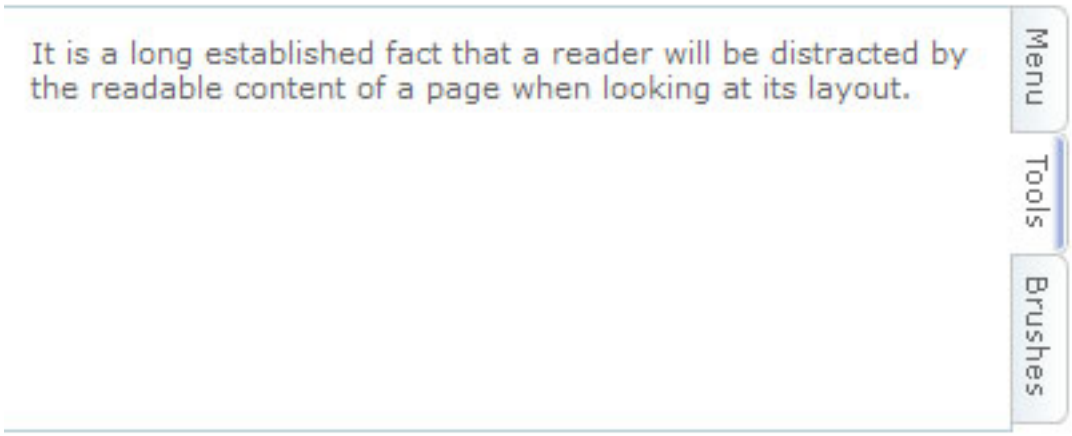
The `b:singular` attribute defines that only one child of an element can be selected at one time. When another child is selected, the other children are automatically deselected.

5.4.5 Sidebarbox

The sidebar control is a tabbed control that you can position vertically on the left-hand side or right-hand side of the screen. A tab is partly visible when inactive, and fully visible when active. You can specify the attributes `b:width`, `b:height`, `b:top`, and `b:side` to indicate whether it appears `right` or `left` (the default). Use the `b:img` attribute to indicate an image file.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.10 - Backbase Sidebarbox


It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.
Menu
Tools
Brushes

The following is an example of a sidebarbox control (to close the control, click the tab that is open):

Example 5.11 - Creating a Sidebar Control

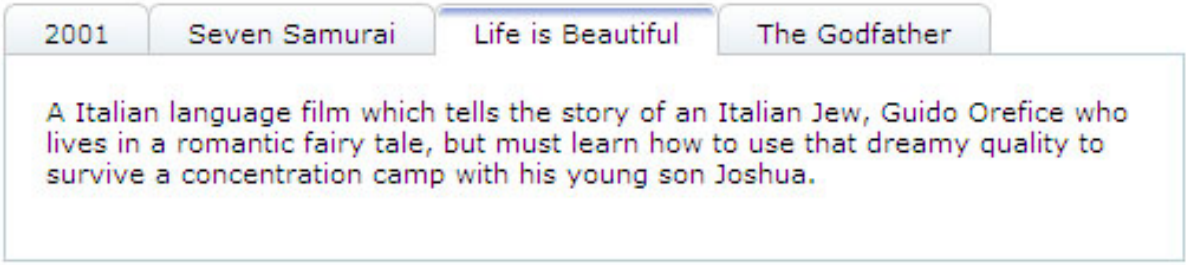
```
<b:sidebarbox b:width="400" b:height="250" b:top="100" b:side="left">
    <b:sidebar b:img="menu.gif">          Lorem Ipsum is simply dummy text of the printing
and typesetting industry....</b:sidebar>
    <b:sidebar b:img="tools.gif">          It is a long established fact that a reader will
be distracted by the readable content of a page when looking at its layout.</b:sidebar>
    <b:sidebar b:img="brushes.gif">          Contrary to popular belief, Lorem Ipsum is not
simply random text. </b:sidebar>
</b:sidebarbox>
```

The images used in the above example are available in the `Backbase\3_1_4\controls\backbase\b-sidebarbox` directory, or in the `3_1_4\basic` directory.

5.4.6 Tabbox

A control that provides a tabbed interface, each tab being defined within a `b:tab` tag. The `b:tabbox` control supports lazy loading: when you specify the tab contents using a `b:url`, the contents are only loaded at the moment a user clicks the tab for the first time.

Figure 5.11 - Backbase Tabbox


2001
Seven Samurai
Life is Beautiful
The Godfather
A Italian language film which tells the story of an Italian Jew, Guido Orefice who lives in a romantic fairy tale, but must learn how to use that dreamy quality to survive a concentration camp with his young son Joshua.

In the following example, four tabs are defined:

Example 5.12 - Creating a Tabbox Control

```
<b:tabbox style="height: 100%">
    <b:tab b:label="2001" b:url="data/2001_short.xml" />
    <b:tab b:label="Seven Samurai" b:url="data/seven_short.xml" />
    <b:tab b:label="Life is Beautiful" b:state="selected" b:url="data/bella_short.xml" />
    <b:tab b:label="The Godfather" b:url="data/godfather_short.xml" />
</b:tabbox>
```

©2004-2006 Backbase B.V., All Rights Reserved.

Where, for example, the `b:url="data/bella_short.xml` contains the following:

Example 5.13 - Response File

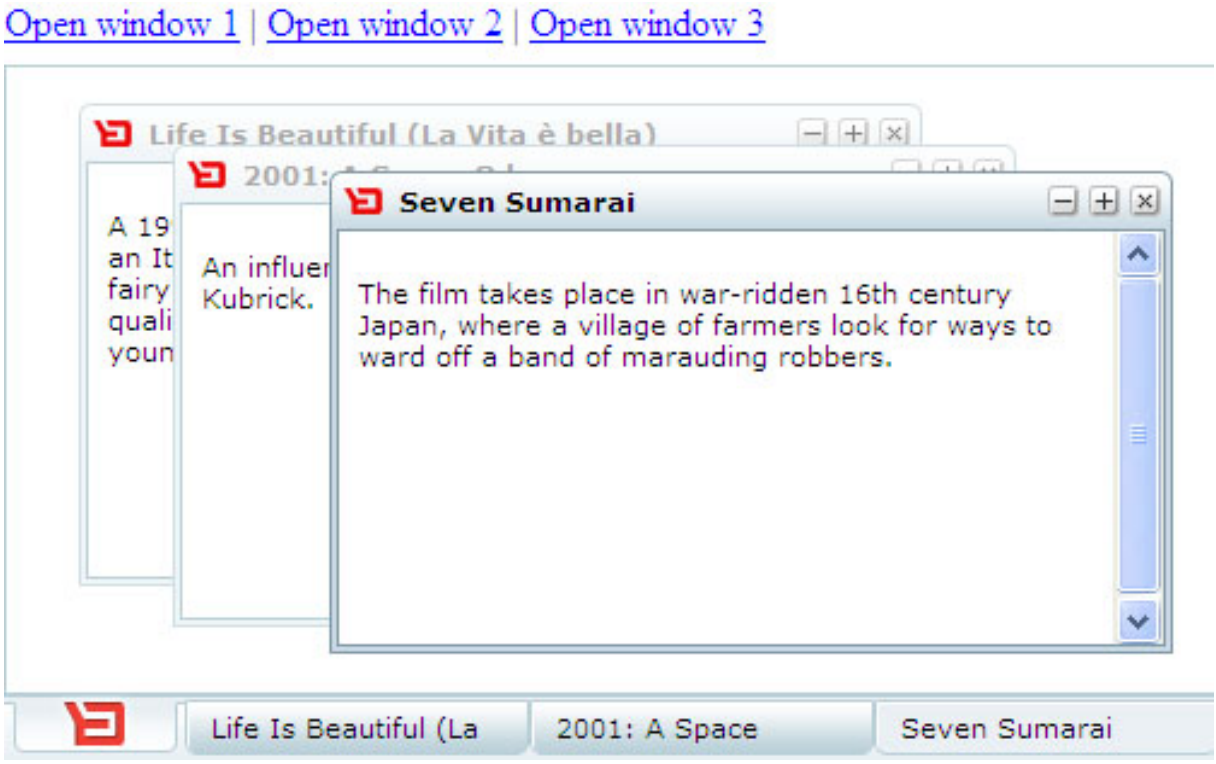
```
<div xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
<h3>Life Is Beautiful (La Vita è bella, 1997)</h3>
<p>      Italian language film which tells the story of an Italian Jew,      Guido Orefice
(played by Roberto Benigni), who lives in a      romantic fairy tale, but must learn how to
     use that dreamy quality to survive a      concentration camp with his      young son
Joshua.</p>
</div>
```

For more information, see *Defining Response Files [p. 47]* and *Lazy Loading [p. 39]* .

5.4.7 Taskbar

The taskbar control is used to manage windows. It creates tabs at the bottom of the screen for each open window.

Figure 5.12 - Backbase Taskbar


Open window 1 | Open window 2 | Open window 3
Life Is Beautiful (La Vita è bella)
Seven Sumarai
The film takes place in war-ridden 16th century Japan, where a village of farmers look for ways to ward off a band of marauding robbers.
Life Is Beautiful (La
2001: A Space
Seven Sumarai

Example 5.14 - Creating a Taskbar Control

```
<b:panelset b:rows="27px * 27px">
    <b:panel>
        <a b:action="trigger" b:event="open" b:target="id('taskbar_win1')">Open window 1</a>
|
        <a b:action="trigger" b:event="open" b:target="id('taskbar_win2')">Open window 2</a>
|
        <a b:action="trigger" b:event="open" b:target="id('taskbar_win3')">Open window 3</a>
    </b:panel>
    <b:panel>
        <b:windowarea style="height: 100%">
            <b:window id="taskbar_win1" b:open="true" style="position: absolute; width:
350px; height: 200px; top:15px; left:30px;">
                <b:windowhead>Life Is Beautiful (La Vita è bella)</b:windowhead>
                <b:windowbody>
                    <p>A 1997 Italian language film which tells the story of an Italian Jew,
Guido Orefice who lives in a romantic fairy tale, but must learn how to
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
                        use that dreamy quality to survive a concentration camp with his
young son Joshua.</p>
                </b:windowbody>
            </b:window>
            <b:window id="taskbar_win2" b:open="true" style="position: absolute; width:
350px; height: 200px; top:25px; left:60px;">
                <b:windowhead>2001: A Space Odyssey</b:windowhead>
                <b:windowbody>
                    <p>An influential science fiction film directed by Stanley Kubrick.</p>
                </b:windowbody>
            </b:window>
            <b:window id="taskbar_win3" b:open="true" style="position: absolute; width:
350px; height: 200px; top:35px; left:90px;">
                <b:windowhead>Seven Sumarai</b:windowhead>
                <b:windowbody>
                    <p>The film takes place in war-ridden 16th century Japan, where a village
of farmers look for ways to ward off a band of marauding robbers.</p>
                </b:windowbody>
            </b:window>
        </b:windowarea>
    </b:panel>
    <b:panel>
        <b:taskbar>
            <b:startbutton />
        </b:taskbar>
    </b:panel>
</b:panelset>
```

5.4.8 Toolbar

The Toolbar control is used to present a navigation bar that can contain images, buttons, and selectors (such as spinner and select box). The elements that are placed in the buttonbar are only limited in height. The buttonbar occupies the full width of an area.

Figure 5.13 - Backbase Toolbar

In the following example, when the page is constructed, the items in the toolbar are created using default images that are available in your Backbase controls directory (the XPath `concat` function is used to find the images relative to the `b:controlpath` set on the `body` tag):

Example 5.15 - Creating a Toolbar Control

```
<xmp b:backbase="true" b:controlpath="..controls/backbase/">
    <div>
        <h1>Toolbar</h1>
        <b:toolbar>
            <b:toolbaritem>
                <img alt="" style="height:14px; width:26px;">
                    <s:event b:on="construct" b:target="@src" b:action="set"
b:value="{concat($bpc_controlpath, 'image_repository/icon/send.gif')}" />
                </img>
            </b:toolbaritem>
            <b:toolbaritem>
                <img alt="" style="height:16px; width:20px;">
                    <s:event b:on="construct" b:target="@src" b:action="set"
b:value="{concat($bpc_controlpath, 'image_repository/icon/compose.gif')}" />
                </img>
            </b:toolbaritem>
            <b:toolbaritem>
```

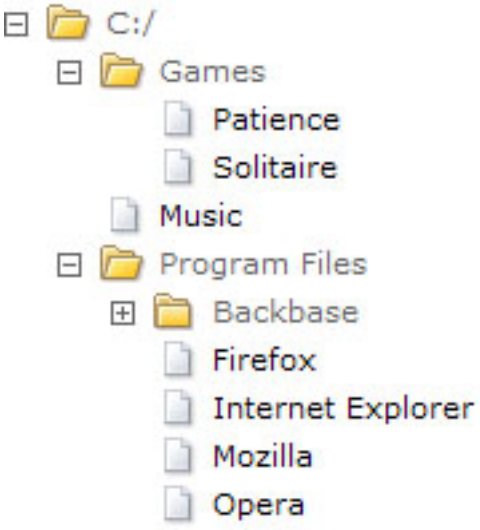
©2004-2006 Backbase B.V., All Rights Reserved.

```
                <img alt="" style="height:14px; width:19px;">
                    <s:event b:on="construct" b:target="@src" b:action="set"
b:value="{concat($bpc_controlpath, 'image_repository/icon/erase.gif')}" />
                </img>
            </b:toolbaritem>
            <b:toolbaritem>
                <img alt="" style="height:16px; width:16px;">
                    <s:event b:on="construct" b:target="@src" b:action="set"
b:value="{concat($bpc_controlpath, 'image_repository/icon/preview.gif')}" />
                </img>
            </b:toolbaritem>
        </b:toolbar>
    </div>
</xmp>
```

5.4.9 Tree

The Tree control creates a simple tree structure. You create the control by simply nesting `b:tree` tags to create the desired tree hierarchy. The `b:tree` has a `b:label` attribute.

Figure 5.18 - Backbase Tree



The following code is an example of a tree control:

Example 5.16 - Creating a Tree Control

```
<b:tree b:label="C:/" b:open="true" style="width: 250px">
    <b:tree b:label="Games">
        <b:tree b:label="Patience" />
        <b:tree b:label="Solitaire" />
    </b:tree>
    <b:tree b:label="Music" />
    <b:tree b:label="Program Files">
        <b:tree b:label="Backbase">
            <b:tree b:label="BPC" />
        </b:tree>
        <b:tree b:label="Firefox" />
        <b:tree b:label="Internet Explorer" />
        <b:tree b:label="Mozilla" />
        <b:tree b:label="Opera" />
    </b:tree>
</b:tree>
```

The `b:tree` control supports lazy loading. For more information, see *Lazy Loading [p. ?]* .

©2004-2006 Backbase B.V., All Rights Reserved.

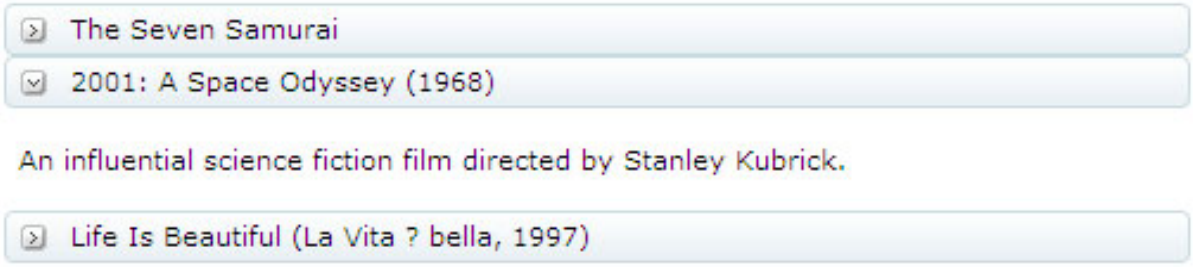
5.5 Defining Information Controls

Use the information controls to provide additional information to the user. You can define information controls using the BXML client controls: `b:accordeon`, `b:infobox`, `s:loading`, `b:statusbar`, and `b:tooltip`.

5.5.1 Accordeon

The Accordeon control groups related information together. It contains head and body elements that define each item in the group. The head element describes the content contained in the body and is always visible; you can expand or collapse the body elements by clicking the head element. Only one of the body elements can be selected at a time; expanding one item automatically collapses the currently selected item (using an animation effect). When selected, the body takes the size of the content.

Figure 5.19 - Backbase Accordeon


The Seven Samurai
2001: A Space Odyssey (1968)
An influential science fiction film directed by Stanley Kubrick.
Life Is Beautiful (La Vita ? bella, 1997)

In the following example, the accordeon contains three accordeonheads and accordeonbodies. You can use the `b:state` attribute to define which of the heads is opened on startup.

Example 5.17 - Creating an Accordeon Control

```
<b:accordeon style="width: 500px">
    <b:accordeonhead>The Seven Samurai</b:accordeonhead>
    <b:accordeonbody>
        <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
    </b:accordeonbody>
    <b:accordeonhead>         2001: A Space Odyssey (1968)</b:accordeonhead>
    <b:accordeonbody>
        <p>An influential science fiction film directed by Stanley Kubrick.</p>
    </b:accordeonbody>
    <b:accordeonhead b:state="selected">         Life Is Beautiful (1997)</b:accordeonhead>
    <b:accordeonbody>
        <p>An Italian language film which tells the story of an Italian Jew, Guido Orefice
(played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use
that dreamy quality to survive a concentration camp with his young son Joshua (played by
Giorgio Cantarini).</p>
    </b:accordeonbody>
</b:accordeon>
```

5.5.2 Loading

The loading control is the only element that is directly loaded by the engine when server interaction takes place to inform the user about what is taking place. When the file finishes loading, the HTML is automatically removed from the document.

Define the loading control as follows:

- You can define your own loading control using the `s:loading` tag to define the HTML to display while the application is loading a response file.
- If you want to use loading with Backbase styling applied, include an `s:include` tag to your startup file using the following syntax:

©2004-2006 Backbase B.V., All Rights Reserved.

```
<s:include b:url="../controls/backbase/s-loading/s-loading.xml" />
```

Where the `b:url` attribute refers to the location of custom control in your Backbase installation 3_0_1 directory.

5.5.3 Infobox

The infobox control provides additional information to the user that they can interact with. The infobox appears when it receives a command to display that information. Infoboxes can contain nested infoboxes that stay open until you click outside the infobox.

Figure 5.20 - Backbase Infobox

Life Is Beautiful
La Vita è bella, 1997
A Italian language film which tells the story of an Italian Jew, Guido Orefice (played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use that dreamy quality to survive a concentration camp with his young son Joshua (played by Giorgio Cantarini).
More info > >

In the following example, the first infobox that appears contains a link to another infobox. The `b:target` attribute define what to trigger with the `select` command. The `b:mode` attribute indicates where the infobox is positioned. Since the infobox does not disappear when you move the mouse, you can interact the content, for example to click the link to the second infobox.

Example 5.18 - Creating an Infobox Control

```
<div>
    <a b:action="select" b:target="id('info1')" b:mode="at-pointer">        Life Is
Beautiful</a>
    <b:infobox id="info1" style="width: 250px">
        <h2>La Vita è bella, 1997</h2>
        <p>A Italian language film which tells the story of an Italian Jew, Guido Orefice
(played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use
that dreamy quality to survive a concentration camp with his young son Joshua (played by
Giorgio Cantarini).</p>
        <a b:action="select" b:target="id('info2')">More info &gt; &gt;</a>
    </b:infobox>
    <b:infobox id="info2" style="width: 250px">
        <h2>La Vita è bella, 1997 (cont'd)</h2>
        <p>Guido tries to comfort Joshua by pretending that their imprisonment is just an
elaborate game. The film also stars Nicoletta Braschi.</p>
    </b:infobox>
</div>
```

5.5.4 Statusbar

The statusbar control is an information bar usually presented at the bottom of an application. \ Specify the postioning of the b:statusbar using CSS, for example `style="position: absolute; bottom:0; left:0; width:100%;"`. The `b:separator /` tag is a divider for use in a statusbar.

Example 5.19 - Creating a Statusbar Control

```
<b:statusbar style="position: absolute; bottom:0; left:0; width:100%;">      Online!
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    <b:separator />
     Offline!
</b:statusbar>
```

5.5.5 Tooltip

A tooltip appears when a user hovers over the element on which it is defined to provide additional information. The user cannot interact with the information, and is closed on a mousedown event.

To create a simple text tooltip, you can add a `b:tooltiptext` attribute to almost any XHTML or BXML tag. For example:

Example 5.20 - Creating a Tooltip Control

```
<p b:tooltiptext="This is a tooltip">Hover Here!</p>
```

To create a more complex tooltip, such as to show images or a Flash movie instead of just plain text, use the `s:tooltip` tag in which you can add XHTML content. For example:

Example 5.21 - Creating a Tooltip Control With An Image

```
<div>
    <a>The Godfather
        <s:tooltip>
            <img src="data/corleone.jpg" />
            <br />
Francis Ford Coppola (1972)
        </s:tooltip>
    </a>
and
    <a>Scarface
        <s:tooltip>
            <img src="data/tony.jpg" />
            <br />
Brian De Palma (1983)
        </s:tooltip>
    </a>
are two of the most important gangster films of the post-war era.
</div>
```

5.6 Defining Chart Controls

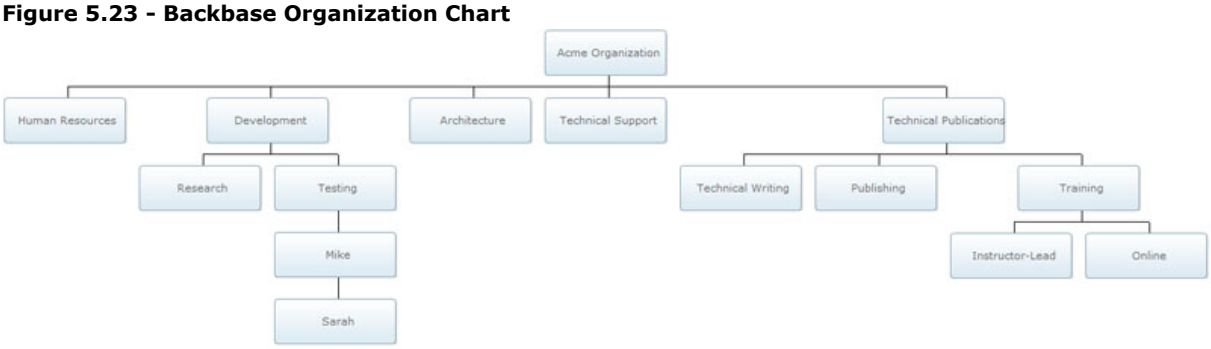
Use the chart controls to present information in a chart. You can define chart controls using the BXML client controls: `b:barchart`, `b:orgchart`, and `b:linechart`.

5.6.1 OrganizationChart

The Organization Chart control is used for defining a hierarchical structure, for example to define an organization. The `b:orgchart` tag describes the top of the hierarchy, and contains `b:subordinate` tags. The `b:subordinate` are nested to create the hierarchy. The `b:label` attribute is a logical String defining the box label.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.23 - Backbase Organization Chart



Example 5.22 - Creating an Organization Chart Control

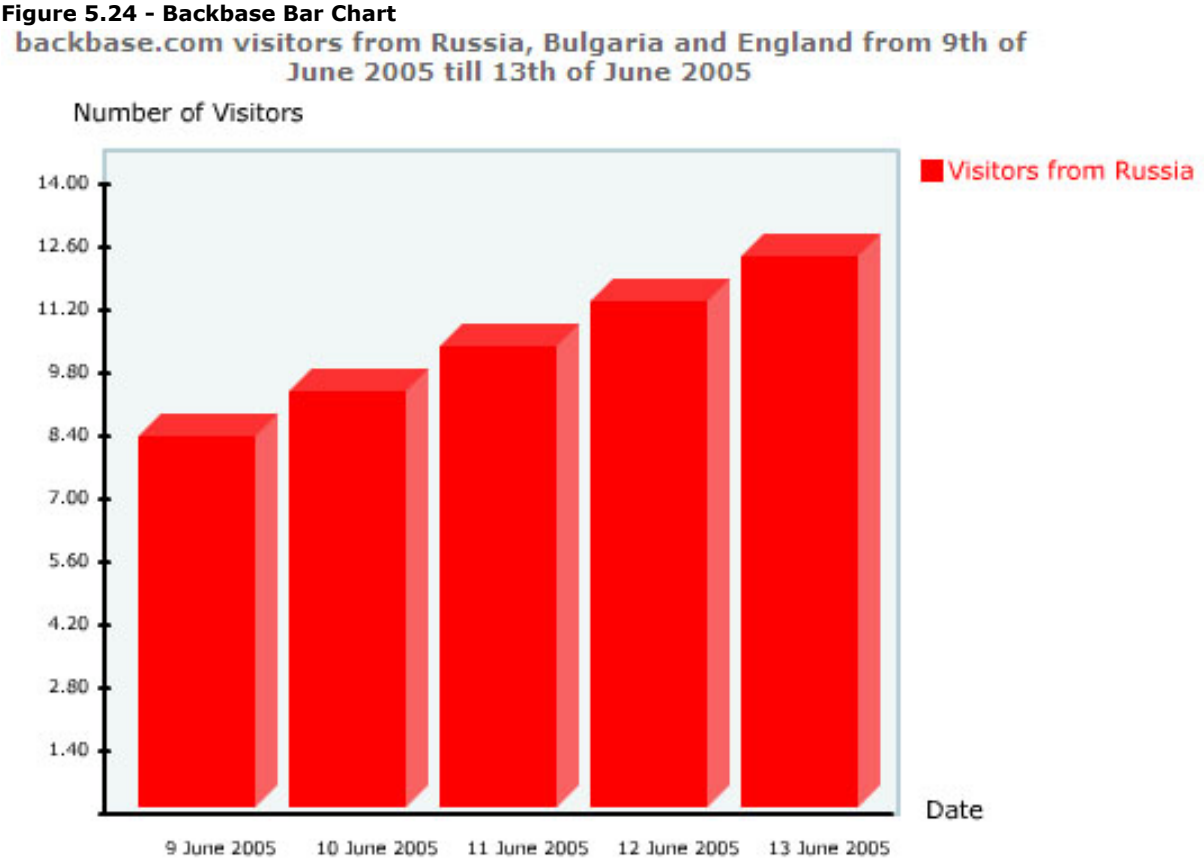
```
<b:orgchart b:label="Acme Organization">
    <b:subordinate b:label="Human Resources" />
    <b:subordinate b:label="Development">
        <b:subordinate b:label="Research" />
        <b:subordinate b:label="Testing">
            <b:subordinate b:label="Mike">
                <b:subordinate b:label="Sarah"></b:subordinate>
            </b:subordinate>
        </b:subordinate>
    </b:subordinate>
    <b:subordinate b:label="Architecture" />
    <b:subordinate b:label="Technical Support" />
    <b:subordinate b:label="Technical Publications">
        <b:subordinate b:label="Technical Writing" />
        <b:subordinate b:label="Publishing" />
        <b:subordinate b:label="Training">
            <b:subordinate b:label="Instructor-Lead" />
            <b:subordinate b:label="Online" />
        </b:subordinate>
    </b:subordinate>
</b:orgchart>
```

5.6.2 Barchart

The barchart control consists of a series of labeled vertical bars that show different values for each bar - it is used to display graphically these differences.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.24 - Backbase Bar Chart



The following is an example of a barchart control:

Example 5.23 - Creating a Backbase Bar Chart

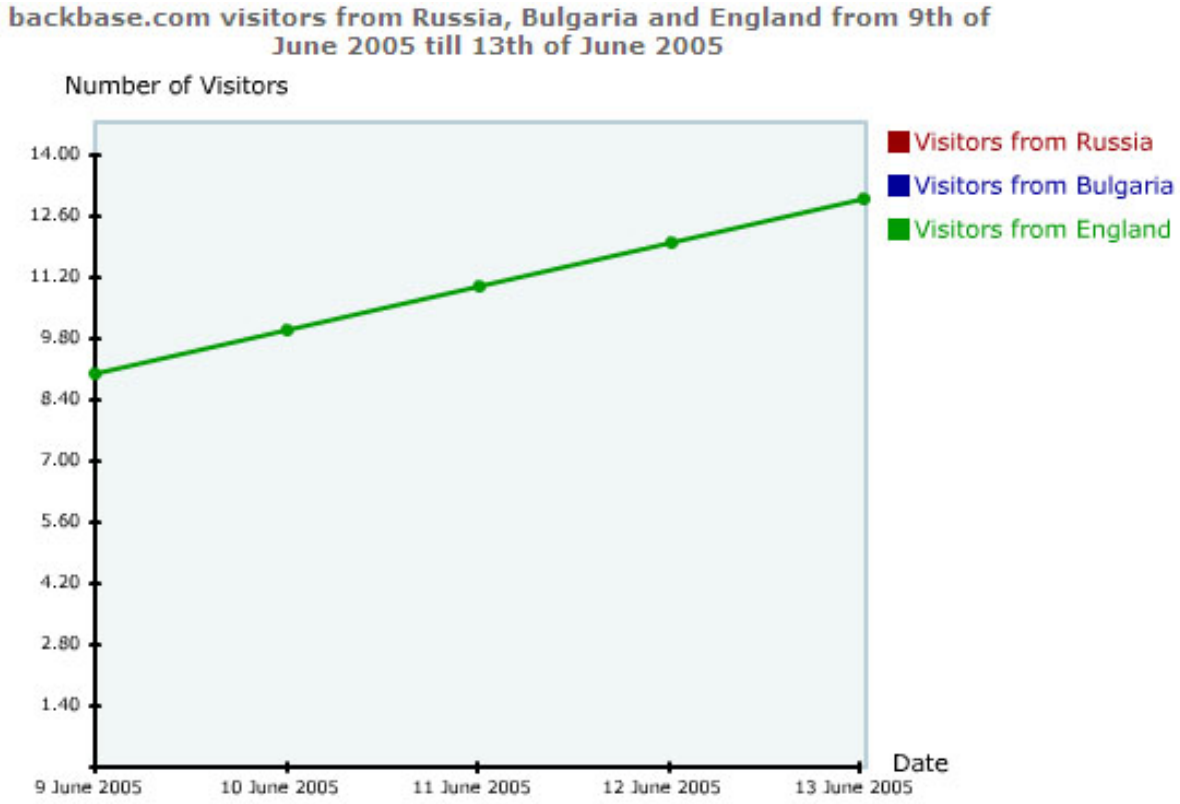
```
<b:barchart b:caption="backbase.com visitors from Russia from 9th of June 2005 till 13th of
June 2005" b:x-axis-label="Date" b:y-axis-label="Number of Visitors" b:name="name"
b:width="600" b:height="400">
    <b:barchart-horizontal-values>
        <b:barchart-value b:value="9 June 2005" />
        <b:barchart-value b:value="10 June 2005" />
        <b:barchart-value b:value="11 June 2005" />
        <b:barchart-value b:value="12 June 2005" />
        <b:barchart-value b:value="13 June 2005" />
    </b:barchart-horizontal-values>
    <b:barchart-series b:name="Visitors from Russia" b:color="ff0000">
        <b:barchart-bar b:value="15" />
        <b:barchart-bar b:value="23" />
        <b:barchart-bar b:value="25" />
        <b:barchart-bar b:value="22" />
        <b:barchart-bar b:value="26" />
    </b:barchart-series>
</b:barchart>
```

5.6.3 Linechart

A line graph is used to show continuing data; how one thing is affected by another. The chart makes it clear to see how things are going by the rises and falls a line graph shows.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.25 - Backbase Line Chart



The following is an example of a linechart control:

Example 5.24 - Creating a backbase Linechart

```
<b:linechart b:caption="backbase.com visitors from Russia, Bulgaria and England from 9th of
June 2005 till 13th of June 2005" b:x-axis-label="Date" b:y-axis-label="Number of Visitors"
b:width="600" b:height="400" b:name="name">
    <b:linechart-horizontal-values>
        <b:linechart-value b:value="9 June 2005" />
        <b:linechart-value b:value="10 June 2005" />
        <b:linechart-value b:value="11 June 2005" />
        <b:linechart-value b:value="12 June 2005" />
        <b:linechart-value b:value="13 June 2005" />
    </b:linechart-horizontal-values>
    <b:linechart-series b:name="Visitors from Russia" b:color="#990000">
        <b:linechart-point b:value="15" />
        <b:linechart-point b:value="23" />
        <b:linechart-point b:value="25" />
        <b:linechart-point b:value="22" />
        <b:linechart-point b:value="26" />
    </b:linechart-series>
    <b:linechart-series b:name="Visitors from Bulgaria" b:color="#000099">
        <b:linechart-point b:value="13" />
        <b:linechart-point b:value="14" />
        <b:linechart-point b:value="12" />
        <b:linechart-point b:value="13" />
        <b:linechart-point b:value="15" />
    </b:linechart-series>
    <b:linechart-series b:name="Visitors from England" b:color="#009900">
        <b:linechart-point b:value="15" />
        <b:linechart-point b:value="20" />
        <b:linechart-point b:value="27" />
        <b:linechart-point b:value="25" />
        <b:linechart-point b:value="25" />
    </b:linechart-series>
</b:linechart>
```

©2004-2006 Backbase B.V., All Rights Reserved.

5.7 Defining Form Controls

Use the form controls to add usability to a form-based user interface. You can define form controls using the BXML client controls: `b:button`, `b:combobox`, `b:datepicker`, `b:selectbox`, `b:slider`, and `b:spinner`.

5.7.1 Button

A button control creates a customizable button element that can contain any XHTML markup.

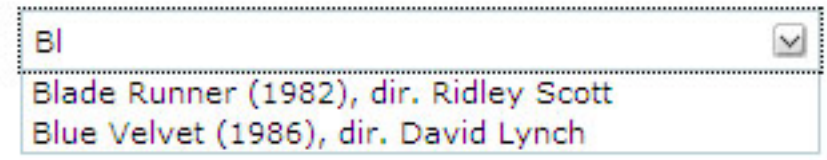
Example 5.25 - Creating a Button Control

```
<b:button b:action="alert" b:value="Hello!">Alert!</b:button>
```

5.7.2 Combobox

The Combobox control is a textarea field with a drop-down listbox containing alternative values from a fixed set. The `b:combobox` tag contains child `b:combo-option` tags defining the list box. The `b:combobox` tag `b:name` attribute is the name of the input field that is sent on a form submit. You can specify an initial value using the `b:value` attribute, or `b:text` to set the initial text that appears in the box. The `b:combobox` also can have the `b:style` attribute to indicate the styling for the input field. The `b:combo-option` tag `b:value` attribute is the value of a value-representation pair.

Figure 5.26 - Backbase Combobox



Example 5.26 - Creating a Combobox

```
<b:combobox b:width="300px" b:name="fieldvalue" b:text="B">
    <b:combo-option b:value="1">2001: A Space Odyssey (1968), dir. Stanley
Kubrick</b:combo-option>
    <b:combo-option b:value="2">A Clockwork Orange (1971), dir. Stanley
Kubrick</b:combo-option>
    <b:combo-option b:value="3">A Hard Day's Night (1964), dir. Richard
Lester</b:combo-option>
    <b:combo-option b:value="4">A Room With A View (1986), dir. James Ivory</b:combo-option>
    <b:combo-option b:value="5">A Streetcar Named Desire (1951), dir. Elia
Kazan</b:combo-option>
    <b:combo-option b:value="6">Adam's Rib (1949), dir. George Cukor</b:combo-option>
    <b:combo-option b:value="7">All About Eve (1950), dir. Joseph Mankiewicz</b:combo-option>
    <b:combo-option b:value="8">All That Jazz (1979), dir. Bob Fosse</b:combo-option>
    <b:combo-option b:value="9">Annie Hall (1977), dir. Woody Allen</b:combo-option>
    <b:combo-option b:value="11">Annie Hall (1977), dir. Woody Allen</b:combo-option>
    <b:combo-option b:value="12">Apocalypse Now (1979), dir. Francis Ford
Coppola</b:combo-option>
    <b:combo-option b:value="13">Blade Runner (1982), dir. Ridley Scott</b:combo-option>
    <b:combo-option b:value="14">Blue Velvet (1986), dir. David Lynch</b:combo-option>
    <b:combo-option b:value="15">Bonnie And Clyde (1967), dir. Arthur Penn</b:combo-option>
    <b:combo-option b:value="16">Bringing Up Baby (1938), dir. Howard Hawks</b:combo-option>
    <b:combo-option b:value="17">Casablanca (1942), dir. Michael Curtiz</b:combo-option>
    <b:combo-option b:value="18">Chinatown (1974), dir. Roman Polanski</b:combo-option>
    <b:combo-option b:value="19">Citizen Kane (1941), dir. Orson Welles</b:combo-option>
    <b:combo-option b:value="19">City Lights (1931), dir. Charles Chaplin</b:combo-option>
</b:combobox>
```

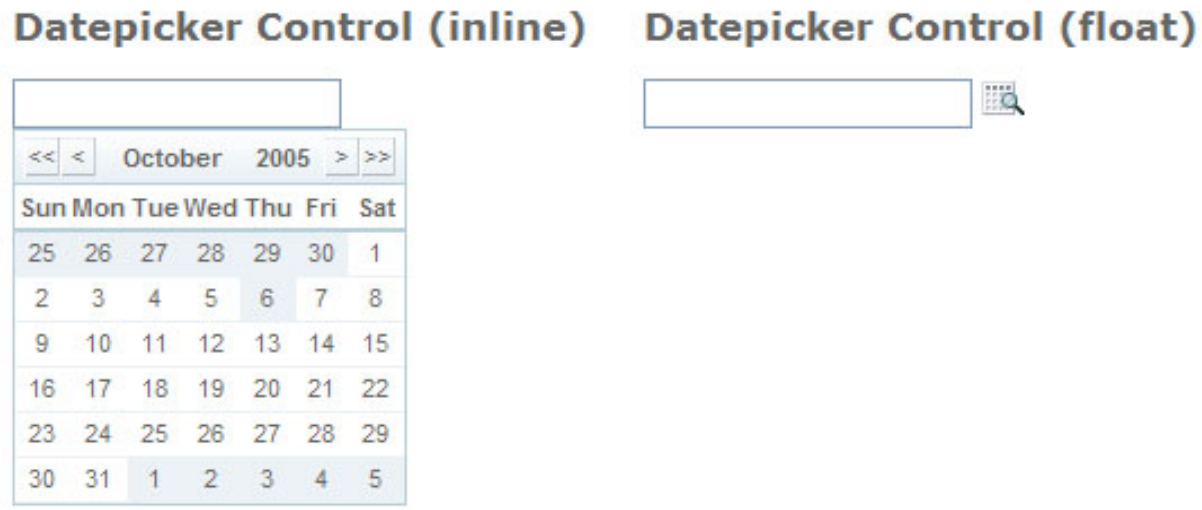
©2004-2006 Backbase B.V., All Rights Reserved.

5.7.3 Datepicker

The datepicker control allows users to select a date from a calendar. The datepicker has the following attributes:

- `b:type` - `inline` displays the calender underneath the element, `float` (the default) displays a clickable calender icon.
- `b:input` - the XPath expression to the input element that is used to display the selected date.
- `b:format` - the date format displayed in the input element, for example `dddd dd MMMM yyyy` results in `Wednesday 05 January 2005`. For more information on date formats, see the `BXML Reference` PDF.
- `b:value` - an initial value for the input element (you must specify a date that conforms to the date format that is specfied in the `b:format` attribute.

Figure 5.27 - Backbase Datepicker



The following examples shows an inline and floating datepicker:

Example 5.27 - Creating a Datepicker Control

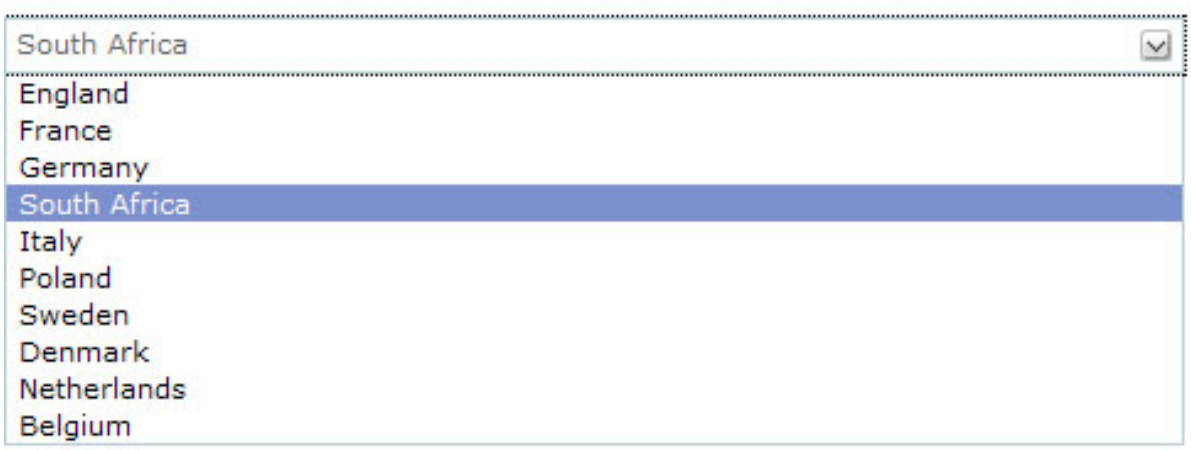
```
<div style="position: absolute;left:20px;">
    <h2>Datepicker Control (inline)</h2>
    <input type="text" name="input1" />
    <b:datepicker b:type="inline" b:input="../input[1]" b:format="dddd dd MMMM yyyy" />
</div>
<div style="position: absolute;left:300px;">
    <h2>Datepicker Control (float)</h2>
    <input type="text" name="input1" />
    <b:datepicker b:input="../input[1]" b:format="dddd dd MMMM yyyy" />
</div>
```

5.7.4 Select

The Select control a drop-down list. You can change the value, when the cursor is in the input field, by pressing the up and down arrow keys, and you can press the spacebar to cancel. The `b:select` tag contains child `b:option` tags that define an item in the list. The `b:select` tag `b:name` attribute is the name of the input field that is sent on a form submit, and the `b:value` attribute sets the initial field value. The `b:option` tag `b:value` attribute is the value of a value-representation pair.

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.28 - Backbase Select



The following is an example of creating a select control:

Example 5.28 - Creating a Select Control

```
<b:select b:name="county" b:value="Sweden" b:width="30%">
    <b:option b:value="1">England</b:option>
    <b:option b:value="2">France</b:option>
    <b:option b:value="3">Germany</b:option>
    <b:option b:value="4">South Africa</b:option>
    <b:option b:value="5">Italy</b:option>
    <b:option b:value="6">Poland</b:option>
    <b:option b:value="7">Sweden</b:option>
    <b:option b:value="8">Denmark</b:option>
    <b:option b:value="9">Netherlands</b:option>
    <b:option b:value="10">Belgium</b:option>
</b:select>
```

5.7.5 Slider

The slider control is a horizontal or vertical gauge that the user can slide within a defined range. The range is defined using the attributes `b:start` and `b:end`, and the attributes. You can give the range a `b:step` value defining increments and decrements. You can handle the change in the slider using the `slide` event.

Example 5.29 - Creating a Slider Control

```
<div style="height:100%">
    <input type="text" b:connect="id('slider1')" value="" />
    <b:slider id="slider1" b:orientation="horizontal" b:start="1" b:end="10" value="5" />
</div>
```

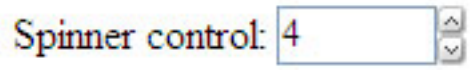
You can set your own image for the control using `s:gfxset`. For more information, see the BXML Reference Manual.

5.7.6 Spinner

The spinner control displays up and down arrows to allow the user to navigate through the values in a value list (usually numbers). If the cursor is in the input field, you can also change the value by pressing the up/down arrow key. You can specify the attributes: `b:start`, `b:value`, `b:step`, `b:end` to specify respectively , and `b:style`

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.29 - Backbase Spinner



Example 5.30 - Creating a Spinner Control

```
<div>     Spinner control:
    <b:spinner b:start="-5" b:value="4" b:step="1" b:end="5" b:style="width: 60px" />
</div>
```

5.8 Defining Data Controls

Use the data controls to manipulate data. You can define data using the BXML client controls: `b:datagrid`, `b:detailviewer`, `b:listview`, `b:listgrid`and `b:treelist`.

5.8.1 Datagrid

Displays cell-based tabular data and supports selecting, sorting, paging, and editing of the data, and has resizable rows and columns. The `b:datagrid` has child elements: `b:datagridhead` and `b:datagridbody`, and `b:datatypes`.

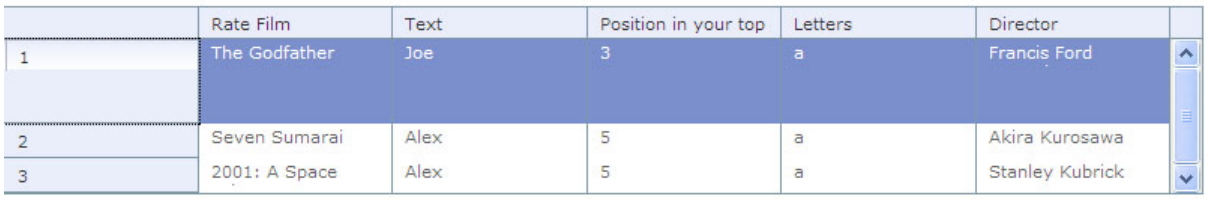
The `b:datagridhead` defines the head and contains a `b:datagridrow` in which the header for each column is defined using a `b:datagridgridheadcell` element.

The `b:datagridbody` defines the body element. The content of the `<b:datagridbody/>` is an HTML table that must be dynamically loaded into the body using the `b:url` attribute. You can specify the following for the `b:url` attribute:

- An XML file that contains raw XML, that is transformed using one of the XSLTs defined in the attributes `b:template` or `b:template-ie5`
- An XML file that contains an HTML table body (attributes and children)
- Any other file type, such as PHP or ASP, that contains and returns raw XML. The XML is transformed using one of the XSLTs defined in the attributes `b:template` or `b:template-ie5`

The `b:datagrid` can also have `b:datatypes` child elements that defines a form element that is applied in the cell of a table. Basically, you can define any HTML or BXML form element that fits within a cell, such as a select box, text input type, `b:spinner`, or `b:datepicker`

Figure 5.30 - Backbase Datagrid



	Rate Film	Text	Position in your top	Letters	Director
1	The Godfather	Joe	3	a	Francis Ford
2	Seven Sumarai	Alex	5	a	Akira Kurosawa
3	2001: A Space	Alex	5	a	Stanley Kubrick

In the following example, the `b:url` specifies an XML file containing raw XML, and uses the XSLTs in the `b:template` and `b:template-ie5` to transform the data:

The files in the `/controls/backbase/b-datagrid` directory also show examples of the datagrid control, including defining contextmenus for row operations; insert, copy, delete, hide, and edit, and XSLT and XML datasource examples.

Example 5.31 - Creating a Datagrid

```
<b:datagrid style="width:800px; height:350px;" b:url="datagrid_film.xml"
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
b:template="xsl/b-datagrid.xsl" b:ie5-template="xsl/b-datagrid-ie5.xsl">
    <b:datagridhead>
        <b:datagridrow>
            <b:datagridheadcell />
            <b:datagridheadcell>Title</b:datagridheadcell>
            <b:datagridheadcell>Director</b:datagridheadcell>
            <b:datagridheadcell>Genre</b:datagridheadcell>
            <b:datagridheadcell>Language</b:datagridheadcell>
            <b:datagridheadcell>Premiere</b:datagridheadcell>
        </b:datagridrow>
    </b:datagridhead>
    <b:datagridbody />
</b:datagrid>
```

The `b:datagrid` control supports page iteration. For more information, see *Implementing a Paging Mechanism [p. 81]*.

5.8.2 DetailViewer

The Detailviewer control presents a detailed view of data such as those used to present technical product details. It presents the data in a table consisting of two columns: a description column and a detail column. Each row of the table is defined using a child `b:property` tag that has a `b:label` attribute to define the text value of the left-hand side column (description).

Figure 5.31 - Backbase Detailviewer

Movie	Finding Nemo
Directors	Andrew Stanton Lee Unkrich (co-director)
Genre	Animation / Comedy / Family
Plot Outline	A father/son underwater adventure featuring Nemo, a boy clownfish, stolen from his coral reef home. His timid father must then search the ocean to find him.

The following is an example of a detailviewer control:

Example 5.32 - Creating a Detailviewer

```
<b:detailviewer>
    <b:property b:label="Movie" b:style="width: 400px">Finding Nemo</b:property>
    <b:property b:label="Directors">Andrew Stanton
        <br />
Lee Unkrich (co-director)
    </b:property>
    <b:property b:label="Genre">Animation / Comedy / Family</b:property>
    <b:property b:label="Plot Outline">A father/son underwater adventure featuring Nemo, a
boy clownfish, stolen from his coral reef home. His timid father must then search the ocean
to find him.</b:property>
</b:detailviewer>
```

The content of `b:property` can be any HTML markup. You can set the width of the contents column (right-hand side column), using the `b:style` attribute. The left-hand column sizes to its contents (`b:label` attributes). To overrule this behavior, you need to change the CSS classes.

5.8.3 Listview

©2004-2006 Backbase B.V., All Rights Reserved.

The listview control presents a sortable table that you can scroll through. The control is different from other controls, in that it is implemented as a behavior; you add the `b:behavior="b-listview"` attribute to an HTML <table> element to make the table fully sortable (click a column head to sort the table according to that column).

Figure 5.32 - Backbase Listview

Movie	Genre	Directory	Language
Apocalypse Now	War	Francis Ford Coppola	English
The Thin Red Line	War	Terrence Malick	English
City of God	Drama	Fernando Meirelles	Portuguese
The Seven Samurai	Drama	Akira Kurosawa	Japanese
Donnie Darko	Drama	Richard Kelly	English
Carlito's Way	Crime	Brian De Palma	English
Scarface	Crime	Brian De Palma	English
The Godfather	Crime	Francis Ford Coppola	English

5.8.4 Listgrid

Displays row-based tabular data, and supports selecting, paging, and editing of the data, and has resizable rows and columns. The `b:listgrid` has child elements: `b:listgridhead` and `b:listgridbody`, and have `b:datatypes`.

The `b:datagridhead` defines the head and contains a `b:datagridrow` in which the header for each column is defined using a `b:listgridheadcell` element.

The `b:listgridbody` defines the body element. The content of the `</b:listgridbody>` is an HTML table that must be dynamically loaded into the body using the `b:url` attribute. You can specify the following for the `b:url` attribute:

- An XML file that contains raw XML, that is transformed using one of the XSLTs defined in the attributes `b:template` or `b:template-ie5`
- An XML file that contains an HTML table body (attributes and children)
- Any other file type, such as PHP or ASP, that contains and returns raw XML. The XML is transformed using one of the XSLTs defined in the attributes `b:template` or `b:template-ie5`

The `b:listgrid` can also have `b:datatypes` child elements that defines a form element that is applied in the cell of a table. Basically, you can define any HTML or BXML form element that fits within a cell, such as a select box, text input type, `b:spinner`, or `b:datepicker`

Figure 5.33 - Backbase Listgrid

Spinner	Text	Numbers	Letters	Date
1	Roger	5	a	2005-12-12
2	Terry	5	a	2005-12-12

In the following example, the `b:url` specifies an XML file containing raw XML, and uses the XSLTs in the `b:template` and `b:template-ie5` to transform the data:

The files in the `/controls/backbase/b-listgrid` directory also show examples of the listgrid control, including defining contextmenus for row operations; insert, copy, delete, hide, and edit, and XSLT and XML datasource examples.

Example 5.33 - Creating a Listgrid

©2004-2006 Backbase B.V., All Rights Reserved.

```
<b:listgrid b:url="datasource.xml" b:template="b-listgrid.xslt"
b:ie5-template="b-listgrid-ie5.xslt">
    <b:listgridhead>
        <b:listgridrow b:width="100">
            <b:listgridheadcell b:width="60px"
b:datatype="spinner">Spinner</b:listgridheadcell>
            <b:listgridheadcell b:width="60px" b:datatype="text">Text</b:listgridheadcell>
            <b:listgridheadcell b:width="60px"
b:datatype="numbers">Numbers</b:listgridheadcell>
            <b:listgridheadcell b:width="60px"
b:datatype="letters">Letters</b:listgridheadcell>
            <b:listgridheadcell b:width="60px" b:datatype="text">Text</b:listgridheadcell>
            <b:listgridheadcell b:width="60px"
b:datatype="datepicker">Date</b:listgridheadcell>
        </b:listgridrow>
    </b:listgridhead>
    <b:listgridbody />
</b:listgrid>
```

The `b:listgrid` control supports page iteration. For more information, see *Implementing a Paging Mechanism [p. 81]*.

5.8.5 Treelist

The Treelist control presents data in a hierarchical list, similar, for example, to a list used in email applications. By default, each row is closed. You can open its children by clicking the row. The `b:treelistrow` defines a row in the `b:treelist` and is displayed like an HTML `<tr>` tag. Within the `b:treelistrow`, you can have a number of `b:treelistcell` tags each defining a column in the treelist row. You can use a `b:treelistchildren` tag as a container to demarcate `b:treelistrow` entries that appear at the same level in the hierarchy.

Figure 5.34 - Backbase Treelist

Subject	From	Posted
Is the Seven Samurai good?	Jeremy Hartley	15-08-2005, 13:01
⊟ 2001 - Question about religion	John Deare	23-07-2005, 17:14
Re: 2001 - Question about religion	Marcel Diepstra	24-07-2005, 8:01
⊟ Vita e Bella Oscar?	John Deare	21-07-2005, 16:07
⊟ Re: Vita e Bella Oscar?	Jeremy Hartley	22-07-2005, 13:27
⊟ Re: Vita e Bella Oscar?	John Deare	22-07-2005, 14:47
Re: Vita e Bella Oscar?	Henry James	23-07-2005, 6:01
Re: Vita e Bella Oscar?	Peter Jones	23-07-2005, 9:43

You can set your own image for the control using `s:gfxset`. For more information, see the BXML Reference Manual.

To use the treelist control, you must add an `s:include` to the path of this control, for example `s:include b:url="../controls/backbase/b-treelist/b-treelist.xml"`

5.9 Control Interaction

5.9.1 Implementing a Paging Mechanism

Paging is essential for large data sets. It allows you to retrieve only a part of the used data set at any time, and to scroll backwards and forwards through the data set page-by-page. You can implement paging for the Backbase data controls such as the `b:datagrid` and `b:listgrid` controls.

©2004-2006 Backbase B.V., All Rights Reserved.

To implement paging for a control, you need to:

- Define the following paging attributes for the control:
  - `b:items-in-total="1000"` - limits the total number of records returned in the data set
  - `b:page-size="20"` - defines the number of records displayed in every page
  - `b:page-number="1"` - defines the initial page that is displayed when the page is constructed
  - `b:page-cache="3"` - defines the number of pages retained in memory (0 or 1 are identical)
  - `b:url="load.php"` - defines the server-side file that implements the business logic to handle the paging attributes
- Define a `b:pager` control and connect it to a data control. The pager control takes care of telling the paged control which page to load, the pager control then makes sure that it shows the correct page control buttons:
  - `b:pager b:observe="id('mytilelist')"` - connects the pager control to the control that requires paging
  - `b:pager-item b:page-type="first"` - defines links to pages in the data set. Possible values are: `first | previous | next | last`
  - `b:pager-numbers b:page-links="5"` - defines how many pages are clickable

  Every child of the paged element has a page number to be able to identify which pages are already loaded. They also have a `b:dirty` attribute to make it possible to force a reload of a specific page.
- Implement the server-side business logic that extracts the records from a data set and splits the records into pages. This requires the synchronization of the client-server control interaction, and the implementation differs depending on your back-end environment:
  - *Backbase .Net Server* and *Backbase Java Server* - synchronization with the server is handled automatically. See the relevant server documentation.
  - PHP/JSP or ASP - synchronization with standard back-ends is handled using the `b:url` attribute and by concatenating the attribute values that are attached as a query string to the url.

The control paging attributes only define what the control wants to receive; you must implement what it actually receives on the server-side implementation. In other words, the control renders all the data it receives from the server.

5.9.2 Lazy Loading

Controls such as `b:tree` and `b:tabbox` support lazy loading. In other words, the content is not loaded before the user actually needs it. To implement lazy loading, you need to use the `b:url` and the `populate` event. The `b:url` attribute is used to send data to the server. The attribute values are concatenated and attached as a query string to the url. A `b:dirty` attribute is used to mark an element for reloading. When the `b:dirty` attribute is set to `false` on an element, the element is not reloaded directly, but only when the information is needed again.

The following code provides an example of how to implement lazy loading:

Example 5.34 - Lazy Loading

```
<s:event b:on="open | select">
    <s:if b:test="@b:dirty = 'true' or (not(@b:dirty) and not(node()))">
        <s:task b:action="trigger" b:event="populate" />
    </s:if>
    <!--Write your control specific open code here-->
</s:event>
<s:event b:on="populate">
    <!--In the BNS and BJS editions you extend this event-->
    <s:task b:test="@b:url" b:action="load" b:url="{@b:url}" />
    <s:setatt b:dirty="false" />
</s:event>
```

©2004-2006 Backbase B.V., All Rights Reserved.

5.10 Adding Look And Feel (CSS)

You can define the look and feel of the Rich User Interface using:

- CSS - CSS skinning can be applied to both XHMTL and BXML. For more information, see W3C standard: `http://www.w3.org/Style/CSS/`
- `s:setstyle` - is used to dynamically assign styles to UI elements in response to events. For example:

  ```
  <s:setstyle b:background-color="green" b:width="200px" b:opacity=".2" b:height="20px" b:top="0px"/>
  ```

- Behavior states - you can use behaviors to define the visual representation of an element when in one of its base states (**selected** or **deselected**). For more information, see *Responding to State Changes (CSS) [p. 109]*
- Add and remove class - use the `addclass` command to add a CSS class to an element (normal CSS priorities apply)

  ```
  <button b:action="addclass" b:value="heading" b:target="//h1" />
  ```

5.11 Using Backbase Styled Controls

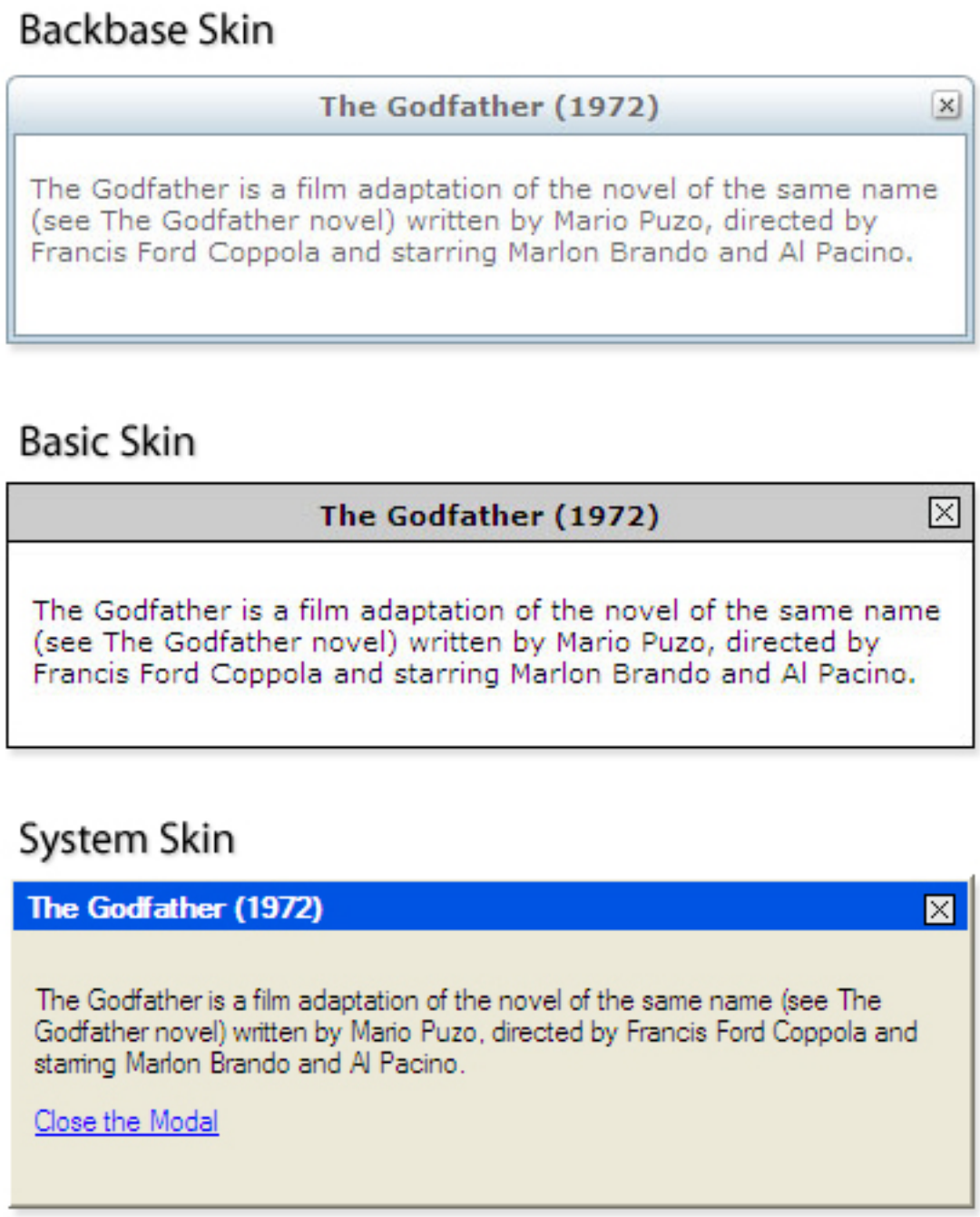
Backbase provides out-of-the-box user interface controls that encapsulate common patterns and behaviors. The Backbase controls are available with different skins (styling). Skins are interchangeable sets of style definitions that allow you to change the web application's look & feel without impacting settings or functionality. Backbase controls are located in your Backbase installation `3_1_4/controls/` directory and are available in the following skins:

- Backbase - applies a "metallic" styling to the controls. The Backbase skin is non-modifiable.
- Basic - applies a simple styling to the controls. The basic skin is modifiable.
- System - applies the styling of your operating system to the controls

The following illustration shows the same `b:modal` control using the different the skins (of course, the system skin differs depending on your operating system styling):

©2004-2006 Backbase B.V., All Rights Reserved.

Figure 5.35 - Basic, Backbase, and System Skins


Backbase Skin
The Godfather (1972)
The Godfather is a film adaptation of the novel of the same name (see The Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon Brando and Al Pacino.
Basic Skin
The Godfather (1972)
The Godfather is a film adaptation of the novel of the same name (see The Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon Brando and Al Pacino.
System Skin
The Godfather (1972)
The Godfather is a film adaptation of the novel of the same name (see The Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon Brando and Al Pacino.
Close the Modal

The styling applies to Backbase controls styles the components of your web application, not the entire web application.

5.11.1 Setting the Control Path

You must set the `b:controlpath` attribute on the `body` tag to indicate the user interface controls you want to use. By default, if you do not specify the `b:controlpath`, the controls contained in the `3_1_4/controls/dynamic/` folder are used. (In a default installation, the `/dynamic` folder contains controls with Backbase skinning applied.)

Set the `b:controlpath` using a relative path to the controls directory, for example:

©2004-2006 Backbase B.V., All Rights Reserved.

Example 5.36 - Controlpath

```
<body onload="bpc.boot('../');" b:controlpath="../controls/backbase/" b:skinbase="none" />
```

You should copy-and-paste the controls you want to use automatically in your application into the folder specified by the `b:controlpath`. You can also remove controls that you are not using to improve performance, and you can also mix the styling used for different controls, for example by using the Backbase skins for some controls or basic styling for others.

It is recommended to create a separate folder in `/controls` directory, for example `/custom/`, in which to create your own custom controls. For more information, see *Backbase Controls [p. 110]*

The controlpath setting has no influence on the menubar and treelist controls. To use these controls, you must add an `s:include` to the path of this control, for example `s:include b:url="../controls/backbase/b-menubar/b-menubar.xml"`

The `b:skinbase` attribute specifies whether the `default.xml` file in the controlpath directory is loaded or not (specify `none` or `default`). The `default.xml` is your control's configuration file that contains default behaviors, styling, and layout for frequently used constructs such as `infobox`, `tooltip`, `selectbox`, and `slider`. It also contains definitions for default validation behavior. It is recommended to use the file as a central location where you can include miscellaneous behavior and styling. Loading the file automatically is a convenient way to ensure your applications default look and feel: the file has an "includes" to the `default.css` located in your controls directory. The file has rules such as:

Example 5.37 - Default CSS

```
* { font-size: 11px; font-family: Verdana,Geneva,Arial,Helvetica,sans-serif; color: rgb(102,
102, 102); }
```

The `default.css` is loaded after the normal CSS includes. To overrule this behavior, set the `b:skinbase="none"` so that the `default.xml`, and therefore the `default.css`, is not loaded.

5.12 Overriding Backbase Styled Controls

You can use the Backbase out-of-the-box controls with Backbase 'metallic' styling, or basic styling applied by specifying the `b:controlpath` attribute.

The **Backbase styling is non-modifiable**, except for positioning styles: `top`, `left`, `bottom`, `right`, `position`, `width`, `height`, and `float`. You can override these properties using inline styles.

The basic styling attributes `font-size`, `font-family`, and `background` are sometimes overruled.

The **basic styling is modifiable** using inline styles and external stylesheets. To overrule the styling, write your own class and add the `-content` suffix to the class name. For example, to change the styling of the Accordeon control:

Example 5.38 - Overruling Basic Controls Styling: Defining

```
.b-accordeonhead-content { height: 22px; width: 100%; padding-left: 3px; background-image:
url(top_left.gif); background-position: 0 0; background-repeat: no-repeat; }
```

Apply the changes as follows:

Example 5.39 - Overruling Basic Controls Styling: Applying

```
<b:accordeonhead class="b-accordeonhead-content">The Seven Samurai</b:accordeonhead>
```

©2004-2006 Backbase B.V., All Rights Reserved.

5.13 Multimedia Elements

You can use any (HTML compatible) multi-media element, such as Audio, Video, and SVG, in BXML tags and in Backbase controls, for example, to use flash animations in your application, use the following syntax:

Example 5.40 - Flash Control

```
<b:flash b:url="tags.swf" b:width="300" b:height="300" />
```

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 6 - Applying User Interactions

6.1 Defining Events and Commands

Events and commands add interactivity to your SPI. Basically, an *event* defines, "when something happens", the command "what actions take place". An event can be a user event or system event:

- **User Event** most events take place when the user does something, such as clicking a link
- **System Event** some events are not directly caused by the user, for example the `construct` event fires when a page has been constructed

You can trigger events for any BXML or XHTML element. You define what happens when an event is triggered in an *event handler*. The event handler handles the event by executing a command, or series of commands. By executing commands you can, for example: load elements, remove elements, change the visual display of elements, change the location of elements, and set attributes. In BXML, commands are declared with the `b:action` attribute. Each command has its own attributes that you need to place on the tag to execute the command. For an overview of the commands you can use, see *Commands [p. 33]* .

The `b:on="..."` attribute specifies the event that is triggered, and the `b:action="..."` attribute specifies the command to execute when the element´s command event fires.

The events supported by Backbase include the standard ones supported by JavaScript, such as `click`, `mouseover`, or `mousenter`. Backbase also introduces other events such as `dragreceive`, `mousedeeper`, and `command`.

Most commands trigger a corresponding event. For example, a `b:action="alert"` command triggers an `alert` event that you can respond to using `b:on="alert"`. However, some commands such as `load` and `send` do not trigger events. For more information, see the Command section of the BXML Reference.

6.1.1 Defining Event Handlers

You can define event handlers in several ways. The following list the ways and why you should use them:

- **Inline** - for any XHTML or BXML element, you can define the `b:action` attribute and specify the command specific attributes.
- **Event Tag** - you can define the `s:event` tag inline, as direct child nodes of XHTML or BXML elements, or you can define the event anywhere in the document, in which case defining a `b:on` attribute in the element triggers the event handler.

  Event bubbling ensures that events are passed to parent elements. For more information, see *Event Bubbling [p. 104]*
- **Behavior Tag** - you can wrap the event handler in a `s:behavior`, in which the element needs to reference the behavior explicitly using the attribute `b:behavior` (the value is the name of the behavior). Behaviors can inherit from other behaviors, and can be used by several elements.

The following are examples defining event handlers:

- On the element itself (`b:action` attribute). It is executed as the event takes place, which means that the event is fired when the user clicks on an element.

```
<a b:action="alert" b:value="Hello">     Click me</a>
```

  When you define a command without an event, the `b:on="command"` event is the default event that is triggered.
- As the `s:task` child element of an event handler which is declared by the `s:event`

©2004-2006 Backbase B.V., All Rights Reserved.

tag.

```
<a>Click me
    <s:event b:on="command">
        <s:task b:action="alert" b:value="Hello" />
    </s:event>
</a>
```

- As the `b:action` attribute of the `s:event` tag. This is a shorthand notation for the example above.

```
<a>Click me
    <s:event b:on="command" b:action="alert" b:value="Hello" />
</a>
```

- As the `s:task` child element of the `s:execute` tag. This special event handler is triggered the moment the newly loaded data is parsed by the BPC runtime. It is similar to `s:event b:on="construct"`.

```
<s:execute>
    <s:task b:action="alert" b:value="Hello" />
</s:execute>
```

- As the `s:task` child element of the `s:if`, `s:when` and `s:otherwise` tags. Tasks within the `s:if` and `s:when` tags are executed only when the test can be satisfied. Tasks within the `s:otherwise` tag are executed if none of the `s:when` tests can be satisfied.

```
<s:choose>
    <s:when b:test="id(@href)">
        <s:task b:action="select" b:target="id(@href)" />
    </s:when>
    <s:otherwise>
        <s:task b:action="load" b:url="{@href}" />
    </s:otherwise>
</s:choose>
```

- Wrapped in a behavior that defines an effect when the event is triggered:

```
<h1 b:behavior="font-resize">Resize Font</h1>
<s:behavior b:name="font-resize">
    <s:event b:on="mouseenter">
        <s:fxstyle b:font-size="5px" />
    </s:event>
    <s:event b:on="mouseleave">
        <s:fxstyle b:font-size="30px" />
    </s:event>
</s:behavior>
```

A behavior can inherit a behavior that has an event handler defined for the **same** event. For more informations, see *Inheriting Behaviors [p. 109]*

6.1.2 Defining Custom Events

You can use the `trigger` command to create a custom event (and also to trigger a system event) using the syntax `b:action="trigger" b:event="myEvent"` (the `b:event` attribute defines the name of the custom event). If you define an event handler for your custom event, for example in a behavior or on the element itself, the event is triggered whenever the `trigger` command is fired.

In the following example, four custom events events are defined. By *capturing keys [p. ?]* with the `s:keys` you can then trigger a custom event when the key is pressed. The event handler for the custom event is defined in a behavior, that results in an alert dialog box appearing when the event is triggered.

Example 6.7 - Custom Events

```
<s:behavior b:name="write">
    <s:event b:on="press_a_key">
        <s:task b:action="alert" b:value="'A' is a hot key!" />
    </s:event>
    <s:event b:on="press_c_key">
        <s:task b:action="alert" b:value="'C' is a hot key!" />
    </s:event>
    <s:event b:on="press_f_key">
        <s:task b:action="alert" b:value="'F' is a hot key!" />
    </s:event>
    <s:event b:on="press_g_key">
        <s:task b:action="alert" b:value="'G' is a hot key!" />
    </s:event>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    <s:whenactive>
        <s:keys b:keys="a" b:action="trigger" b:event="press_a_key" b:target="id('dest')" />
        <s:keys b:keys="c" b:action="trigger" b:event="press_c_key" b:target="id('dest')" />
        <s:keys b:keys="f" b:action="trigger" b:event="press_f_key" b:target="id('dest')" />
        <s:keys b:keys="g" b:action="trigger" b:event="press_g_key" b:target="id('dest')" />
    </s:whenactive>
</s:behavior>
<div style="position: absolute;left:20px; top:55px;" id="dest" b:behavior="write">
    <textarea style="width: 300px; height: 100px">Start writing and find the active
keys!</textarea>
</div>
```

The code example in the section *Locking For Explicit Synchronization [p.]* provides another examples of defining a custom event.

6.1.3 Input Events

The following table describes the user input events you can use in Backbase:

Event	Description
active	Take places on an element when it, or a child node, gets focus.
blocked-mousedown	Takes place instead of a normal mousedown event when the left mouse button is pressed on an element that is blocked from receiving input events.
blur	Takes place when a field loses focus.
focus	Takes place when a field gets focus.
change	Takes place when the content of an input field has changed.
child-lost	Takes place on the parent of an element which has been successfully dragged-and-dropped, or moved.
click	Takes place when the left mouse button is clicked over an element.
command	Takes place when an element is selected by user input (mouse or keyboard).
dblclick	Takes place when the left mouse button is clicked twice over an element.
drag-deeper	Takes place when an element that is being dragged is moved from a parent element to a child element (down the BXML tree hierarchy). It is triggered on the parent element.
drag-drop	Takes place when an element is succesfully dropped on a receiving element. It is triggered on the element that is being dropped.
drag-enter	Takes place when the element that is being dragged is moved from the parent element to a child element (down the BXML tree hierarchy). It is triggered on the element

©2004-2006 Backbase B.V., All Rights Reserved.

Event	Description
	which the dragged element enters.
`drag-leave`	Takes place when the element that is being dragged is moved from a child element to a parent element. It is triggered on the child element.
`drag-receive`	Takes place on the element which receives the element that has been dragged and dropped.
`drag-reenter`	Takes place when the element that is being dragged is moved from the child element to a parent element (up the BXML tree hierarchy). It is triggered on the element which the dragged element reenters.
`drag-start`	Takes place when the drag action commences. It is triggered on the element that is being dragged.
`inactive`	Take places on an element when it, or a child node, loses focus.
`mousedeeper`	Takes place the cursor moves over the children of an element.
`mousedown`	Takes place an element receives the mouse pressed down input.
`mouseenter`	Takes place when an element received the mouse over input.
`mouseleave`	Takes place when an element receives the mouse out input.
`mouseup`	Takes place when an element received the mouse unpressed input.
`mousereenter`	Takes place when the pointing device's cursor moves out of the children of an element (the cursor leaves the children of the element's box).
`receive`	Takes place when an element receives an element as a result of a `copy`, `move`, or `render` actions.
`resizewindow`	Takes place when the browser window is resized (triggered on the `body` tag).
`rmbclick`	Takes place when a user clicks an element with the right mouse button.
`rmbdown`	Takes place when an element receives the right mouse button pressed down input.
`rmbup`	Takes place when an element received the right mouse button unpressed input. Note: You must set the `b:bubble` attribute to false on the `body` tag disactivate the standard browser pop-up menus.

©2004-2006 Backbase B.V., All Rights Reserved.

Event	Description
`scrollwindow`	Takes place when the browser window is scrolled.
`slide`	Takes place when the user moves the thumb of a slider control.

Event bubbling ensures that events are passed to parent elements. Event bubbling only applies to input events. For more information, see *Event Bubbling [p. 104]*

6.1.4 System Events

The following table describes the system events you can use in Backbase:

Event	Description
`close`	Takes place when a node (a `b:treelistrow`) in a treelist is closed.
`construct`	Takes place when the system is done with parsing an element.
`deselect`	Takes place when an element is deselected.
`disable`	Takes place when an element is disabled (an element, once disabled, cannot respond to user input events, and can only be enabled using a system event.)
`enable`	Takes place when an element is enabled (an element, once disabled, cannot respond to user input events, and can only be enabled using a system event.)
`load`	Takes place when a document is loaded and parsed by the BPC.
`open`	Takes place when a node in a treelist or menu is opened.
`remove`	Takes place when element is destroyed by a remove command.
`select`	Takes place when an element is selected.
`submit`	Takes place when a form is submitted.

6.2 Sequence of Event Triggering

This section describes the initial execution order of events for: **left-mouse double-click**, **right-mouse double-click**, **drag-and-drop**, and **focus and blur**.

It is important to remember that if a user action causes multiple events to be triggered, they are triggered asynchronously. To force the events to execute synchronously, that is one-by-one, use the `s:lock` tag and target an element that serves as the locking element.

©2004-2006 Backbase B.V., All Rights Reserved.

6.2.1 Left-Mouse Button Double-Click

When a left-mouse button double-click action takes place, the event sequence is as follows:

1. mousedown
2. mouseup
3. command
4. click
5. mousedown (not in IE)
6. mouseup (not in IE)
7. command (not in IE)
8. click (not in IE)
9. dblclick

6.2.2 Right-Mouse Button Double-Click

When a right-mouse button double-click action takes place, the event sequence is as follows:

1. rmbdown
2. rmbup
3. rmbclick
4. rmbdown
5. rmbup
6. rmbclick
7. rmbdblclick

6.2.3 Drag-And-Drop

When a drag-and-drop action takes place, the event sequence is as follows:

1. drag-start (if an element is received)
2. child-lost
3. drag-drop
4. drag-receive

6.2.4 Element Receives Focus

When an element receives focus, the event sequence is as follows (the element that loses focus gets a blur event:

1. blur
2. focus

6.3 Adding User Interaction Events

Users interact with the interface using the mouse and keyboard. Backbase has events that are triggered in reaction to a user's mouse action, for example mouseover, mousedown, and click, and you can intercept key presses from the keyboard, and bind actions to them, using the `s:key` tag.

6.3.1 Mouse Events

The following example demonstrates mouse events. When a mouse event takes place, a behavior is used to temporarily activate the list item corresponding to the mouse action that occurs within the `div` element.

Example 6.8 - Defining Mouse Events

©2004-2006 Backbase B.V., All Rights Reserved.

```
<div style="border: 1px solid #BBB; width: 250px; height: 100px;">
    <s:event b:on="click" b:action="select" b:target="../ul/li[1]" />
    <s:event b:on="dblclick" b:action="select" b:target="../ul/li[2]" />
    <s:event b:on="mouseenter" b:action="select" b:target="../ul/li[3]" />
    <s:event b:on="mouseleave" b:action="select" b:target="../ul/li[4]" />
    Interact with this area to show the mouse events...
</div>
<ul>
    <li b:behavior="mouse-active">Click</li>
    <li b:behavior="mouse-active">Double click</li>
    <li b:behavior="mouse-active">Area enter</li>
    <li b:behavior="mouse-active">Area leave</li>
</ul>
<s:behavior b:name="mouse-active">
    <s:state b:on="select" b:normal="mouse-active" />
    <s:state b:on="deselect" b:normal="" />
    <s:event b:on="select">
        <s:fxstyle b:time="500" />
        <s:task b:action="deselect" />
    </s:event>
</s:behavior>
<style type="text/css">     .mouse-active {     color: red;     font-weight: bold;
    }</style>
```

For more information on events, see *Input Events [p. 89]* .

6.3.2 Capturing Keys

You can use the `s:keys` tag to map a command to a key, or keys. The `s:keys` tag requires the attributes, `b:keys` and `b:action`. (Any additional attributes form the arguments for the command.) The value of the `b:keys` attribute can be the key character (case is ignored), or the decimal ASCII value of the key. For example, `27` is the **Esc** key. The `b:action` defines the command that is executed when the key(s) is pressed.

The `s:whenactive` tag is a wrapper for the `s:keys` tag to provide the application context for when the keys are active. The keys are only active when the elements to which they are applied are active. You can define keys inline, or inside a behavior (but they must always be contained in a `s:whenactive`). For example, to define global keys, you can define them in a `s:behavior` that is used in the `body` element to be active throughout your application.

The following is a simple example of how to capture keys. When the first textarea is active, pressing the a or z keys toggles the element with the id `keys`:

Example 6.9 - Defining Keys

```
<div style="position: absolute;left:20px; top:55px;">
    <textarea style="width: 300px; height: 100px">Press a or z keys to toggle a textarea
(when active)</textarea>
    <s:whenactive>
        <s:keys b:keys="a" b:action="show" b:target="id('keys')" />
        <s:keys b:keys="z" b:action="hide" b:target="id('keys')" />
    </s:whenactive>
</div>
<div id="keys" style="position: absolute;left:20px; top:105px;">
    <textarea style="width: 300px; height: 100px">Pressing a or z keys shows and hides this
element</textarea>
</div>
```

For more information about keys, see *Changing the Default Navigation Keys [p. 98]* and *Changing the Tools Configuration [p. 26]* .

The following tables lists some keyboard keys and their corresponding ASCII value:

Key	Value
Esc	27

©2004-2006 Backbase B.V., All Rights Reserved.

Key	Value
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
Left Arrow	37
Up Arrow	38
Right Arrow	39
Down Arrow	40

6.4 Defining Focus Model

The focus model allows you to define the elements in your user interface that a user can navigate through using keys. An element that has focus can respond to keyboard input. By default, HTML form elements are focusgroups and focusitems.

You define elements that can get focus by placing the following BXML attributes, with the values `true` or `false`, on an element. The attributes define a hierarchy of focus elements in the BXML space:

- `b:focusroot` - define the main focus container(s). You can have multiple focus roots inside your application, but there is no higher level.
- `b:focusgroup` - define a grouping of items that can get focus and that you can navigate through.
- `b:focusitem` - identify an element that can get focus. An element that has focus can respond to keyboard input. The `_focusCurrent` variable contains the focusitem that currently has focus (is null when no element has focus).

 The attributes define a hierarchy of focus elements. You can nest all elements as long as the hierarchy is observed. For example, a focusroot can have nested focusroots, a focusgroup can have nested focusgroups, and focusitem nested focusitems. A focusgroup, or a focusroot, can also be a focusitem.

Default keys are assigned to the focusroot, focusgroup, and focusitem elements to allow you to navigate forwards and backwards through these elements (implemented using commands):

Command	Default Key
`focusitem-next`	Arrow Keys: `right` and `down`

©2004-2006 Backbase B.V., All Rights Reserved.

Command	Default Key
focusitem-previous	Arrow Keys: left and up
focusgroup-next	tab
focusgroup-previous	shift+tab
focusroot-next	ctrl+right
focusroot-previous	ctrl+down
focus-celldown	Table command to focus the same column in the next row (no default)
focus-cellup	Table command to focus the same column in the previous row (no default)

Note: Focus commands **do not** trigger events of the same name.

The following example demonstrates visually the focus model. The focus model elements change color to indicate the state changes that are triggered as you navigate through the elements:

Example 6.10 - Focus Model

```
<s:behavior b:name="focusobject">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:background-color="#0099FF" />
        <!--purple-->
    </s:event>
    <s:event b:on="active">
        <s:setstyle b:background-color="#00D856" />
        <!--green-->
    </s:event>
    <s:event b:on="inactive">
        <s:setstyle b:background-color="#D85500" />
        <!--red-->
    </s:event>
    <s:event b:on="focus">
        <s:setstyle b:background-color="white" />
    </s:event>
</s:behavior>
<div b:behavior="focusobject" b:focusroot="true">
    <p>         This document demonstrates the backbase focus model. It contains two
focusgroups, 1 and 2. Focusgroup 1 contains focusitems 1.1 and 1.2. Focusgroup 2 contains
focusitems 2.1, 2.2, 2.3 and 2.4. You can set focus with the
        <b>keyboard</b>
or the
        <b>mouse</b>
.
    </p>
    <p>The tab and arrow keys are used for keyboard navigation:</p>
    <ul>
        <li>Use the
            <b>Tab</b>
, or
            <b>Tab+Shift</b>
, key to navigate between groups (the focusgroup remembers the focusitem that last had focus
within a group.)
        </li>
        <li>Use the up and down
            <b>arrow keys</b>
change focus between items within a group.
        </li>
    </ul>
    <p>         The focus system is hierarchical. Navigate through the groups and items and
watch how the color indicates all focus containers upwards in the hierarchy. When the page is
constructed, the color
        <b>Purple</b>
is applied as the background. When events are triggered, the elements change color:
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <b>Green</b>
indictates actived elements,
        <b>Red</b>
indicates inactived elements, and
        <b>White</b>
indicates the item that has focus (and is also active):
    </p>
    <div b:behavior="focusobject" b:focusgroup="true">Focusgroup 1
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
1.1</div>
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
1.2</div>
    </div>
    <div b:behavior="focusobject" b:focusgroup="true">Focusgroup 2
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
2.1</div>
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
2.2</div>
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
2.3</div>
        <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
2.4</div>
    </div>
</div>
```

The following is a screenshot of the code example:

Figure 6.1 - Focus Model

6.4.1 Focus Events

The events and commands for the focus model allow you to respond to state changes in elements that are focusable:

- `focus` and `blur`

  The `focus` and `blur` events take place when a user navigates through focusitems

©2004-2006 Backbase B.V., All Rights Reserved.

in your focus model. The `focus` event takes place on the focusitem that gets focus, and a `blur` event takes place on the focusitem that no longer has focus (is blurred).

- `active` and `inactive`

  An element can have one of the states active or inactive: An element is active when it or a child node has focus; and an element is inactive when it or a child node does not have focus (is blurred). The `active` and `inactive` events take place on elements when they change their state.

- `enable` and `disable`

  An element in the focus model (focus root, group, or item) can be enabled (the default), or disabled (by setting the `b:disabled="false"` attribute). The `enable` and `disable` events takes place when an element's enables or disabled state changes. (No user input events can be triggered on an element that is disabled, only system events).

  You can change the value of the attribute using the `s:setatt` tag (use the `s:with` tag to set attributes on elements other than the current), for example:

```
<s:event b:on="disable">
    <s:setatt b:disabled="true" class="input-standard" />
</s:event>
```

  You can also use the `set`, or `remove` commands for changing or removing attribute values.

Note: Event bubbling applies to the events in the focus model. Event bubbling ensures that events are passed to parent elements, although you can block these events being triggered using the `b:eventblock="false"` attribute. For more information, see *Event Bubbling [p. 104]*.

6.4.2 Implementing the Focus Model in the HTML Space

You can implement the focus model for tables in the HTML space by adding the attribute `b:focustype="htmltable"` to the `b:xhtml` tag. Each cell of the table is then a focusitem that can get focus using the keyboard arrow keys. The `bpc_focusCurrentElement` variable contains the focusitem in the HTML space that currently has focus (is null when no element has focus), and the `bpc_focusLastElement` variable contains the focus item that previously has focus (is set when the `blur` event is fired).

The following example shows a table defined within the HTML space. Note also the `focus` event is triggered when the focus item is selected:

Example 6.12 - Focus Model in HTML Space

```
<s:behavior b:name="focusvariables">
    <s:event b:on="focus" b:action="alert" b:value="{concat ('The current focus element is ',
$_focusCurrentElement, ' the previous was ', $_focusLastElement)}" />
</s:behavior>
<b:xhtml b:focusitem="true" b:focusindicator="true" b:focustype="htmltable"
b:behavior="focusvariables">
    <table cellpadding="2" border="1">
        <thead bgcolor="#9acd32">
            <tr>
                <td>Heading 1</td>
                <td>Heading 2</td>
                <td>Heading 3</td>
                <td>Heading 4</td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>A1</td>
                <td>A2</td>
                <td>A3</td>
```

©2004-2006 Backbase B.V., All Rights Reserved.

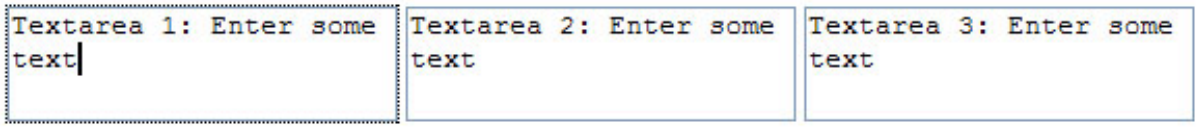
```
                <td>A4</td>
            </tr>
            <tr>
                <td>B1</td>
                <td>B2</td>
                <td>B3</td>
                <td>B4</td>
            </tr>
            <tr>
                <td>C1</td>
                <td>C2</td>
                <td>C3</td>
                <td>C4</td>
            </tr>
        </tbody>
    </table>
</b:xhtml>
```

6.4.3 Changing the Default Navigation Keys

To override the default keys that are used to navigate through focusroots, focusgroups, and focusitems, you can use the `s:whenactive` tag in which you define `s:keys` elements, a wrapper for defining keys (the `b:keys` attribute>) and the command they trigger. You can use the tag inline, or inside a behavior.

In the following example, three text areas are defined that use a behavior called `focus1`. The behavior defines that when a focusitem that uses the behavior gets focus, that the user cannot use the tab and tab+shift keys (these are disactivated by not coupling an action to them) to navigate between text areas (all form elements are automatically focusgroup and focusroot elements). Instead, you must use the left and right arrow keys.

Figure 6.2 - Focus Model



Example 6.13 - Disabling Focusgroups

```
<s:behavior b:name="focus1" b:focusitem="true" b:focusindicator="true">
    <s:whenactive>
        <s:keys b:keys="right" b:action="focusgroup-next" b:bubble="true" />
        <s:keys b:keys="left" b:action="focusgroup-previous" b:bubble="true" />
        <s:keys b:keys="tab" b:bubble="false" />
        <s:keys b:keys="tab+shift" b:bubble="false" />
    </s:whenactive>
</s:behavior>
<div style="position: absolute;left:50px; top:50px;">
    <textarea b:behavior="focus1">Textarea 1: Enter some text</textarea>
    <textarea b:behavior="focus1">Textarea 2: Enter some text</textarea>
    <textarea b:behavior="focus1">Textarea 3: Enter some text</textarea>
</div>
```

The `b:focusindicator` attribute visualizes the focused element by creating a dotted line around the element. In the example above, the focus model is implemented but would not be visible without `b:focusindicator` set to `true`.

The following example demonstrates how you can implement the `focus-cellup` and `focus-cellup` commands in a table, and how you can override the default keys:

Example 6.14 - Table Cell Up and Down Keys

```
<s:include b:url="../controls/backbase/b-listview/b-listview.xml" />
<s:behavior b:name="focus" b:focusitem="true" b:focusindicator="true">
    <s:whenactive>
        <s:keys b:keys="up" b:action="focus-cellup" b:bubble="true" />
        <s:keys b:keys="down" b:action="focus-celldown" b:bubble="true" />
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    </s:whenactive>
</s:behavior>
<div focusroot="true">
    <table b:behavior="b-listview" style="position: absolute;left:20px; top:25px;">
        <thead>
            <tr>
                <th>Movie</th>
                <th>Genre</th>
            </tr>
        </thead>
        <tbody b:focusgroup="true">
            <tr>
                <td b:behavior="focus">The Seven Samurai</td>
                <td b:behavior="focus">Drama</td>
            </tr>
            <tr>
                <td b:behavior="focus">City of God</td>
                <td b:behavior="focus">Drama</td>
            </tr>
            <tr>
                <td b:behavior="focus">Donnie Darko</td>
                <td b:behavior="focus">Drama</td>
            </tr>
            <tr>
                <td b:behavior="focus">2001: A Space Odyssey</td>
                <td b:behavior="focus">Drama</td>
            </tr>
        </tbody>
    </table>
</div>
```

You can navigate through the table rows and columns using the arrow keys:

Figure 6.3 - Focus Model: Table Commands and Keys

6.5 Drag-And-Drop and Resizing

Drag and drop, and resize are two extremely useful and widely used ways for users to manipulate UI elements with the mouse. You can make almost any element draggable and resizable with BXML. For example, you can drag and drop table rows, `div` elements, inline span elements, list items, and form elements. In Backbase, these are implemented declaritively as follows:

- Set the `b:drag` attribute to make an element draggable.
- Set the `b:draggroup` attribute to make a group of elements draggable via one of its elements .
- Set the `b:dragreceive` attribute to make an element a receiver to drop draggable elements.
- Set the `b:resize` attribute to make an object resizable.

You can use Backbase variables within event handlers defined for drag-and-drop functionality. For more information, see *Backbase Variables [p.]*

6.5.1 Dragging and Dropping Elements

To make an element draggable, add a `b:drag` attribute to the element and enter a logical

©2004-2006 Backbase B.V., All Rights Reserved.

String identifier as its value.

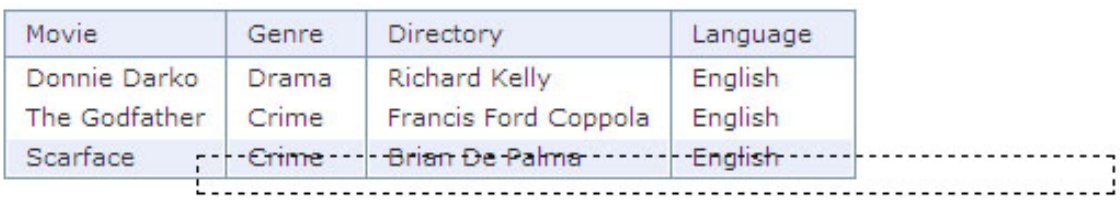
In the following example, you can drag-and-drop the entire table, and also drag-and-drop each row in the table to reorganize the table, or to drag elements from one table to another. The `b:dragmode` indicates the appearance of the dragged element. The `b:dragreceive` the elements that can receive draggable elements (the value is the logical String identifier of the element that can be recieved).

Example 6.15 - Drag-And-Drop

```
<table b:behavior="b-listview" b:drag="outline-example" b:dragmode="real" style="position:
absolute; top: 30px; left: 30px;" width="400">
    <thead>
        <tr>
            <th>Movie</th>
            <th>Genre</th>
            <th>Directory</th>
        </tr>
    </thead>
    <tbody b:dragreceive="a">
        <tr b:drag="a" b:dragmode="outline-none">
            <td>The Seven Samurai</td>
            <td>Drama</td>
            <td>Akira Kurosawa</td>
        </tr>
        <tr b:drag="a" b:dragmode="outline-hidden">
            <td>Donnie Darko</td>
            <td>Drama</td>
            <td>Richard Kelly</td>
        </tr>
        <tr b:drag="a" b:dragmode="outline-none">
            <td>The Godfather</td>
            <td>Crime</td>
            <td>Francis Ford Coppola</td>
        </tr>
        <tr b:drag="a" b:dragmode="outline-hidden">
            <td>Scarface</td>
            <td>Crime</td>
            <td>Brian De Palma</td>
        </tr>
    </tbody>
</table>
<table b:behavior="b-listview" style="position: absolute; top: 150px; left: 30px;"
width="400">
    <thead>
        <tr>
            <th>Movie</th>
            <th>Genre</th>
            <th>Directory</th>
        </tr>
    </thead>
    <tbody style="height: 10px" b:dragreceive="a">
        <tr />
    </tbody>
</table>
```

The `b:dragmode outline-hidden` indicates that the dragged element is immediately removed from its containing element:

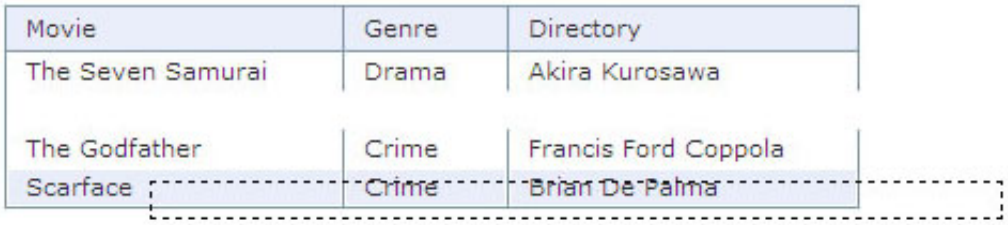
Figure 6.4 - Hidden Drag



©2004-2006 Backbase B.V., All Rights Reserved.

The `b:dragmode outline-none` indicates that an empty space occupied by the dragged element remains in the containing element until the drop is completed:

Figure 6.5 - None Drag



The `b:draggroup` also identifies a draggable element. You can use it make an element such as a window draggable and specify the element, such as the head, that it can be dragged by. You indicate the gripping point for dragging by setting the `b:drag` attribute on the element with the same identifier as the `b:draggroup` attribute.

An element receiving draggable elements can be empty, but it must have its own space in the BXML tree.

You can place drag-and-drop elements in relative, static, or absolute positions:

- **Relatively positioned** or **static elements** must be dropped into a receiving group, otherwise they are return to their starting position. When they are dropped into the receiving group, they snap to the appropriate position within the receiving group's flow. If they are larger than the receiving group they will cause the receiving group to resize.
- **Absolutely positioned elements** remain where they are being dropped even if they are not dropped into a receiving group. When dropped into a receiving group, absolutely positioned elements do not incorporate themselves into the receiving element's flow.

Set `b:overflowfix="true"` on the `b:draggroup` element to prevent Internet Explorer from crashing when using `b:dragmode="real"`.

6.5.2 Resizing Elements

You can resize any relatively or absolutely positioned element by adding the `b:resize` attribute to it. You can add the `b:resize` attribute to virtually any XHTML tag or B-tag. When you move your mouse to the edges of an element with the `b:resize` attribute added, the cursor changes into a resize cursor to indicate the element is resizable. Hold your mouse button down and move it in the desired direction to resize the element.

As it is not always desirable for an element to be unlimitedly resizable, you can set the `b:minwidth` and `b:minheight` attributes to set the minimum width and height of an element in pixels, for example `200px`.

The value of the `b:resize` attribute indicates the directions in which you can resize the element:

- `all`: top, bottom, left and right side.
- `nw`: top and left side only.
- `n`: top side only.
- `ne`: top and right side only.
- `e`: right side only.
- `se`: bottom and right side only.
- `s`: bottom side only.
- `sw`: bottom and left side only.

©2004-2006 Backbase B.V., All Rights Reserved.

- `w`: left side only.
- `horizontal`: left and right side only.
- `vertical`: top and bottom side only.

The code below shows examples of resizable objects. Try also setting the attribute `b:resizemode` to the values `line`, `outline`, or `real`.

Example 6.16 - Resizing Elements

```
<div b:resize="all" class="MyClass">You can resize this border &lt;div&gt; IN ALL
DIRECTIONS</div>
<div b:resize="vertical" b:minheight="200px" class="MyClass">You can resize this border
&lt;div&gt; VERTICALLY</div>
<div b:resize="horizontal" b:minwidth="200px" class="MyClass">You can resize this border
&lt;div&gt; HORIZONTALLY</div>
<style type="text/css">      .MyClass {      position:relative;      border: 2px solid #000;
     background: #00C0C0;      margin: 10px;      padding: 10px;      font-family: Verdana;
     font-size: 10px;      }</style>
```

The following list presents additional functionality and information about resizing:

- **Limiting resizing:** you can limit the resizing of an element using the attributes: `b:minwidth`, `b:minheight`, `b:maxwidth`, and `b:maxwidth`, and you can specify an XPath query which where the element cannot be resize outside of `b:resizeconstraint`.
- `b:resizemode="line"`: when you specify the attribute `b:resizemode="line"`, you can also use the `b:lineconstraint` attribute to specify an element, using XPath, from which the line gets its height or width from.
- `$bpc_resizeLine`: you can use this XPath variable to specify the styling which needs to be applied when line-resizing (by default, it is set to `border:4px solid #AAA;`.
- **Panelset resizing:** the default `b:resizemode` for panelset resizing is `line`. You can override this by specifing the `b:resizemode` on the panelset element you want to resize, for example you can set the bottom and top panels to respectively `b:resizemode="real"` and `b:resizemode="outline"`. You can resize panels with percentages, em and px. A panel specified with * cannot be resized. When you resize percentages, the percentages are maintained.
- **Table resizing:** when resizing tables, it is recommended to place in every `td` and `div` element an `overflow:hidden` to prevent the column from being resized smaller than the content inside. You must also not have padding on the `th` or `td` because this cannot be resized and can give inaccurate results.

  `b:columnmode` is an attribute that only applies when you resize a table: by default, the table grows larger and smaller as you resize the collumns. When you specify `b:columnmode="fixed"`, the table becomes fixed and when you resize a column, only the column next to it grows smaller or larger.

In some situations, resizing can be used incorrectly where resizing is not strictly possible. For example, trying to resize the following elements results in strange behavior: non-blocklevel elements, non- relative or absolute positioned elements, resize north of a non- absolute element, and resizing sides of an element that are fixed.

6.6 Adding Event Propagation

Event propagation allows you to pass events from one UI element to another. You propagate events in Backbase as follows:

- `b:followstate` - allows you to define relationships and synchronize the states of different UI controls. The observer element follows the state (selected or deselected) of the observed element.
- `b:observe` - passes all events triggered on the observed element to the observer element. The `b:observe` is a more generic implementation of `b:followstate`, which can only follow the state (selected or deselected) of the observed element.
- `Event bubbling` - ensures that events are passed to parent elements. You can

©2004-2006 Backbase B.V., All Rights Reserved.

disable this behavior by setting the attribute `b:bubble="false"`
- `b:eventblock` - deactivate events on elements when needed

6.6.1 State

At any one time, an element in a Backbase application can be in a state **selected**, or in a state **deselected**. You can specify the default state of an element using the `b:state` attribute (specify a value `selected` or `deselected`. Most BXML tags show a visible reaction according to their state, for example a modal is active and visible when it is in the selected state, and is hidden when it is in the deselected state.

You can use behaviors to define the visual representation of an element when in one of its base states (**selected** or **deselected**). For more information, see *Behaviors [p. 109]*.

If an element is triggered by a `select` action while the element already is in the selected state, or vice versa, the element event handler is not triggered. For example:

Example 6.17 - Setting State

```
<div b:state="selected">
    <s:event b:on="select" b:action="show" />
    <s:event b:on="construct" b:action="select" />
</div>
```

6.6.2 b:followstate

Followstate allows you to synchronize elements based on state changes. You can add the `b:followstate` attribute to an element (the observer) to observe and react to another element's (the observed) state changes. The `b:followstate` attribute value is an XPath expression pointing to the observed element. The observer element follows the state (selected or deselected) of the observed element.

In the following example, the link has an event handler defined to change its state from selected to deselected each time the link is clicked. The `b:navbox` elements follow the state of the link, therefore clicking the link opens and closes the navboxes:

Example 6.18 - Followstate

```
<div>
    <a id="openclose" b:cursor="pointer">Open/Close All Boxes</a>
    <s:event b:on="click">
        <s:task b:action="select-deselect" b:target="id('openclose')" />
    </s:event>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>The Seven Samurai</b:navhead>
        <b:navbody>
            <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune. The
film takes place in war-ridden 16th century Japan.</p>
        </b:navbody>
    </b:navbox>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>2001: A Space Odyssey</b:navhead>
        <b:navbody>
            <p>An influential science fiction film directed by Stanley Kubrick.</p>
            <a href="http://kubrickfilms.warnerbros.com/">More Info</a>
        </b:navbody>
    </b:navbox>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>Life Is Beautiful</b:navhead>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <b:navbody>
            <p>A Italian language film which tells the story of an Italian Jew who uses his
fantastical imagination to survive a                concentration camp with his young son
Joshua.</p>
        </b:navbody>
    </b:navbox>
</div>
```

6.6.3 b:observe

The `b:observe` attribute passes all events triggered on the observed element to the observer element. The value of the `b:observe` attribute is an XPath expression targeting the observed element. The observer element observes all the events of the observer element. For the observer element to respond to an observed element event, you need to create an event and prefix the event name with `observe-`, for example `b:on="observe-command"`. The `b:observe` is a more generic implementation of `b:followstate`.

Example 6.19 - Observe

```
<p id="observed" b:cursor="pointer">
    <s:event b:on="command">
        <s:task b:action="show-hide" b:target="*[2]" />
    </s:event>
    <div>Click here to show and hide the
        <em>observed</em>
element
    </div>
    <div style="font-weight: bold; color: #04435E;">This is the observed element</div>
</p>
<p b:observe="id('observed')" b:cursor="pointer">
    <s:event b:on="observe-command">
        <s:task b:action="show-hide" b:target="*[2]" />
    </s:event>
    <div>The
        <em>Observing</em>
element mimics the behavior of the observed element.
    </div>
    <div style="font-weight: bold; color: #CE0000;">...Observing Element</div>
</p>
```

6.6.4 b:eventblock

Setting the `b:eventblock` attribute to `true` on an element disables all user input events on it and its children. You can use the `b:eventblock` attribute to create modal dialogs where only the modal dialog responds to user input. You can overrule the eventblock by using the `blocked-mousedown` event.

Example 6.20 - Blocking Events

```
<div b:eventblock="true">
    <s:event b:on="blocked-mousedown" b:action="alert" b:value="Please Close the Dialog!" />
    <a>Click me</a>
</div>
```

6.6.5 Event Bubbling

Event bubbling ensures that when an event is triggered, that the event is passed to parent elements. For example, when you click a button, the click event is triggered. If an event handler is defined for that element for the click event, it is triggered. The click event then bubbles through the object hierarchy until it reaches the top-most level (the document object) and triggers all event handlers defined for the `click` event, unless you set a attribute `b:bubble="false"`.

In the following example, an `b:on="click"` event handler is defined within the `div`

©2004-2006 Backbase B.V., All Rights Reserved.

id="parent" element. When a user clicks anywhere inside the div, for example in a nested div element, the click event "bubbles" up the hierarchy looking for an event handler for the click event. Event bubbling therefore occurs on all objects except for the div element which contains the s:event tag extended with the b:bubble="false" attribute. Clicking any of the other elements triggers an alert box with the text 'Event bubbling!'. The link Three has an additional click event handler defined. It is triggered first before event bubbling takes place.

Example 6.21 - Event Bubbling

```
<div id="parent">
    <s:event b:on="click" b:action="alert" b:value="Event bubbling!" />
    <div>
        <a>One</a>
    </div>
    <div>
        <a>Two</a>
        <s:event b:bubble="false" b:on="click" />
    </div>
    <div>
        <a>Three</a>
        <s:event b:on="click" b:action="alert" b:value="Local alert" />
    </div>
</div>
```

Event bubbling applies to user input events such as click, command, and mouseover, but not for events such as select and show. For more information, see *User Input Events [p. 89]* .

6.7 Adding Effects

A transition is a visual effect that is split up into a number of steps from start to finish in such a way that it appears animated. A transition is formed by one or more s:fxstyle tags with one or more effects. These tags define how the elements that use them should visually react to the events that trigger the effects. Transitions are placed in event handlers or tasklists, extending a given element with additional behavioral properties.

Example 6.22 - Defining Effects

```
<s:behavior b:name="info-trans-example">
    <s:event b:on="select">
        <s:fxstyle b:height="100%" />
    </s:event>
    <s:event b:on="deselect">
        <s:fxstyle b:height="0px" />
    </s:event>
</s:behavior>
```

You can create transition effects as follows:

- Color and CSS Transitions
- Move
- Motion
- b:action="fxtile"

6.7.1 Color and CSS Transitions

You can use any CSS style that can be expressed numerically, such as background-color or font-size, to create a transition effect. CSS styles which cannot be expressed numerically, such as font-family or background-attachment, cannot be used as an s:fxstyle, as it is not possible to calculate the space that intervenes two states. The s:setstyle tag. and s:fxstyle tags are similar; except with the former the transition is immediate, and with the latter the transition takes place in a number of steps over a period of time, which can range from a few hundred milliseconds to several minutes even.

In the following example, a transition effect is created for the background colors. You ap-

©2004-2006 Backbase B.V., All Rights Reserved.

ply the behavior to an element, using the syntax `<tr b:behavior="fx-color">`. When a `mouseenter` event takes place, the background color of the element gradually changes to the color `#7d8fce` over the course of 250 milliseconds. When a `mouseleave` event takes place, the background color of the element gradually changes to the color `#7d8fce` over the course of 1000 milliseconds.

Example 6.23 - Defining Color Transitions

```
<s:behavior b:name="fx-color">
    <s:event b:on="mouseenter">
        <s:fxstyle b:background-color="#7d8fce" b:time="250" />
    </s:event>
    <s:event b:on="mouseleave">
        <s:fxstyle b:background-color="#ffffff" b:time="1000" />
    </s:event>
</s:behavior>
```

6.7.2 Controlling Transitions

You can specify the following attributes to control a transition effect:

- `b:time`
  Indicates, in milliseconds, how long an effect takes to complete. Depending on the amount of time given, the transition will be completed in more or less steps. The number of steps it takes to complete the transition is not solely determined by the amount of time given however, it is also determined by the processor speed of the machine which runs the browser that is performing the transition. The faster the machine, the smoother the transition will appear.
- `b:motion`
  Determines how the individual steps of the transition relate to one another. In a linear transition (`b:motion="linear"`), all steps of the transition are of equal size, so the motion appears smooth and sequential. A logarithmic transition (`b:motion="log"`), on the other hand, starts gradually and increases in size with each step. This causes the motion to be very slow at first, and very rapid at the end. You can also use your own custom JavaScript function to determine how the motion occurs.
- `b:cancel`
  Determines what happens when two contradictory effects are activated at the same time. For example, imagine that a button has an event handler for a `mouseenter` event which starts a color transition. If the same button has another event handler for a `mouseleave` event, which starts an opposing color transition, then you can have two conflicting transitions if the user moves the mouse cursor enters and leaves the button quickly. By default, `b:cancel` is `true`: the first transition is immediately aborted when the second transition is triggered. If the `b:cancel` is set to `false`false, then the first transition is allowed to complete before the second transition is started.

6.8 Capturing Browser History and Bookmarking

In a Multi Page Interface (MPI) web site, each page has a unique URL. These URLs can be used to bookmark your location, and the browsing history is just a list of those URLs in the order you visit them. An AJAX application uses a Single Page Interface (SPI) that has a single URL (contents are dynamically included into the same page) making it difficult for the user to bookmark a specific location. Bookmarks are based on a page navigation concept.

In an AJAX application, the visible content of a page is determined dynamically based on the user input, and this does not always correspond to simply navigating to new content. For example, in an application where an overview of products is shown based on the user filtering criteria, products hidden when a user applies a filter as opposed to separate pages being shown To update the URL on the fly to bookmark a location in a Single Page

©2004-2006 Backbase B.V., All Rights Reserved.

Interface, you have to use the fragment identifier. The fragment identifier is the string that appears after the # mark, and this part can be updated dynamically without reloading the page. When the fragment identifier is updated, the browser also adds this to its history, so the same mechanism automatically supports the back and forward buttons. Based on this fragment identifier, the application can determine how to act.

You can capture browser history in Backbase powered web applications using the `s:history` and the `b:name="browser"` attribute to program the browser Back and the Forward navigation buttons using events and tasks. For example:

Example 6.24 - Browser History

```
<s:behavior b:name="history">
    <s:event b:on="select">
        <s:history b:name="browser">
            <s:task b:action="select" />
        </s:history>
    </s:event>
</s:behavior>
```

When the `s:history` is processed, the tasks it contains are added to the browser history list resulting in a updated URL containing the fragment identifier, for example `#[0]`, `#[1]`, and so on (every new history item increments the number).

Bookmarking in an SPI application requires you to identify state changes in your AJAX site to locate the best positions to add history items. You can then create bookmarks by adding a `b:bookmark` attribute to the `s:history` tag. For example:

Example 6.25 - Bookmarking

```
<s:history b:name="browser" b:bookmark="{@id}">
    <s:task b:action="select" />
</s:history>
```

The contents of the `b:bookmark` attribute are used for the bookmark part of the URL. The attribute can contain a String, but usually it is an XPath expression. For example, `b:bookmark="{@id}"` uses XPath to take the ID from the element the `s:history` tag has been applied to. (You can construct more advanced bookmarks using the concat XPath function.)

In the following example, you can bookmark each tab in the tabbox. When you click the tabs, note that the URL changes, for example `...#SevenSamurai[2]`, allowing you to bookmark it. Each tab uses a behavior called history. When a tab is selected, the `s:history` tag creates a new history item for the tab, using the `id` attribute to bookmark the tab. The `b:name="browser"` attribute enables the user to navigate back and forward using the browser history buttons.

The `b:on="construct"` event is triggered when the application is (re)loaded. When you have bookmarked for example the tab called Seven Samurai, the tab is selected using the ID the bookmark points to. The value of the bookmark is accessed using the `$bpc_bookmark` XPath variable. (The value only contains the bookmark, and no other parts of the URL such as the [1] history suffix. Also, instead of selecting, you can do other things as well, such as loading a file from a path the bookmark points to.)

The files used in this example are available in your Backbase installation `explorer/data` directory

Example 6.26 - Adding Bookmarks

```
<s:event b:on="construct" b:action="select" b:target="id($bpc_bookmark)" />
<s:behavior b:name="history" b:behavior="b-tab">
    <s:event b:on="select">
        <s:super />
        <s:history b:name="browser" b:bookmark="{@id}">
            <s:task b:action="select" />
        </s:history>
    </s:event>
</s:behavior>
<b:tabbox style="height: 100%">
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    <b:tab b:label="2001" id="2001" b:url="data/2001_short.xml" b:behavior="history" />
    <b:tab b:label="Seven Samurai" id="SevenSamurai" b:url="data/seven_short.xml"
b:behavior="history" />
    <b:tab b:label="Life is Beautiful" id="LifeisBeautiful" b:state="selected"
b:url="data/bella_short.xml" b:behavior="history" />
    <b:tab b:label="The Godfather" id="llama" b:url="data/godfather_short.xml"
b:behavior="history" />
</b:tabbox>
```

Note: The `b:behavior="b-tab"` attribute and the `s:super` tag in this example are there to make sure the tabbed navigation control works properly, they are not related to history or bookmarks. If you want to have back-button history without bookmarks, simply use the s:history without b:bookmark attribute.

6.8.1 Creating Custom History Navigation

You can create your own history by specifying a name other than `browser` for the `s:history b:name` attribute. You must then program the your own Back and the Forward navigation buttons using the `forward` and `backward` commands. For example, replace the behavior above with the following:

Example 6.27 - Adding Custom History

```
<s:behavior b:name="history" b:behavior="b-tab">
    <s:event b:on="select">
        <s:super />
        <s:history b:name="myHistory">
            <s:task b:action="select" />
        </s:history>
        <s:task b:action="bookmark" b:value="{@id}" />
    </s:event>
</s:behavior>
<s:event b:on="construct" b:action="select" b:target="id($bpc_bookmark)" />
<div>
    <b:button b:action="backward" b:history="myHistory">&lt; Previous</b:button>
    <b:button b:action="forward" b:history="myHistory">Next &gt;</b:button>
</div>
```

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 7 - Formalizing Patterns

7.1 Behaviors

A *behavior* is a generic construct used to encapsulate functionality for reuse, and to separate the structure of the document from the behavior. You can use behaviors to do the following:

- Define the visual representation of an element when in one of its base states, **selected** or **deselected**, using the `s:state` tags
- Define the instructions (tasks and commands) that are executed when an event takes place (using the `s:event` tag)
  For more information on events, see *Defining Event Handlers [p. 87]*
- Define the keys that active within an element `s:whenactive`
- Set the initial attributes for an element using `s:initatt`

In BXML, you define a behavior using the `s:behavior` tag. The name of the behavior, for example `s:behavior b:name="mybutton"`, is used to assign the behavior to a UI element, for example `b:button b:behavior="mybutton"`. Behaviors can inherit from other behaviors allowing you to create a hierarchy of frequently used events and commands.

7.1.1 Responding to State Changes (CSS)

You can use behaviors to define the visual representation of an element when in one of its base states (**selected** or **deselected**). When in either of the states, you can specify, with a CSS class, how the element appears during the following mouse states:

- `b:normal` - the style class to use when the element is in its base state.
- `b:hover` - the style class to use when the mouse hovers over the element (corresponds to CSS :hover and mouseover event).
- `b:press` - the style class to use when the element is being pressed (corresponds to the CSS :active and mousedown event).
- `b:disabled` - the style class to use when the element is disabled

For example, you can assign CSS classes to each state of the element to change the look & feel of an element as follows:

Example 7.1 - Defining State Behaviors

```
<div b:behavior="statebehavior">     Click or hover here to see the
    <code>s:state</code>
tag in action.
</div>
<s:behavior b:name="statebehavior">
    <s:state b:on="deselect" b:normal="sb" b:hover="sb sb-hover" b:press="sb sb-press" />
    <s:state b:on="select" b:normal="sb sb-selected" b:hover="sb sb-selected sb-hover"
b:press="sb sb-selected sb-press" />
    <s:event b:on="command" b:action="select-deselect" />
</s:behavior>
<style type="text/css">      .sb {      border: 2px solid #889;      padding: .3em;
cursor: pointer;      }      .sb-hover {      background: red;      }      .sb-selected {
     background: blue;      color: white;      }      .sb-press {      background: yellow;
    }</style>
```

The CSS classes that are applied to an element using the `s:state` tag are concatenated to existing classes applied to an element.

7.1.2 Inheriting Behaviors

A behavior can inherit a behavior that has an event handler defined for the **same** event. The event after the `s:super` tag is only processed after the execution of the inherited behavior.

In the following example, the behavior called `movedown` defines an event handler for

©2004-2006 Backbase B.V., All Rights Reserved.

moving objects down in the object hierarchy. It inherits the `zoom` behavior that handles the visual style changes. The `s:super` tag makes sure that the event of the same type in the inherited behavior is applied before starting to run the core behavior instructions for that event.

Example 7.2 - Behavior Inheritance

```
<s:behavior b:name="zoom">
    <s:event b:on="click">
        <s:fxstyle b:font-size="15px" />
        <s:fxstyle b:background-color="#B8B8B8" />
        <s:fxstyle b:font-size="20px" />
        <s:fxstyle b:background-color="#888888" />
    </s:event>
</s:behavior>
<s:behavior b:name="movedown" b:behavior="zoom">
    <s:event b:on="click">
        <s:super />
        <s:task b:action="movedown" b:target="." />
        <s:task b:action="show-hide" b:target="span[@at='moveddown']" />
    </s:event>
</s:behavior>
<div>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">Life is
Beautiful &gt;&gt;
        <span at="moveddown">is an Italian language film which tells the story of an Italian
Jew who lives in a romantic fairy tale, but must learn how to use that dreamy quality to
survive a concentration camp with his young son Joshua.</span>
    </p>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">2001: A Space
Odyssey &gt;&gt;
        <span at="moveddown">is an influential science fiction film notable for combining
episodes contrasting high levels of scientific and technical realism with transcendental
mysticism.</span>
    </p>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">The Seven
Samurai &gt;&gt;
        <span at="moveddown">...takes place in war-ridden 16th century Japan, where a village
of farmers look for ways to ward off a band of marauding robbers.</span>
    </p>
</div>
```

You can place the `s:super` tag as a child element of an `s:event`, `s:parallel` or `s:sequential` tag.

7.2 User-Defined Controls

You can use the `s:htmlstructure` tag to create your own controls. Using HTML structures improves the implementation of declarative interfaces by supplying a central definition of user interface controls and HTML code sequences. You can then use these in your web application using custom tags (the `b:name` attribute of the `s:htmlstructure` tag).

The Backbase controls, located in the `C:\Backbase\3_1_4\controls\backbase` directory, provide examples of how to implement this tag.

The child nodes of the `s:htmlstructure` tag belong to the HTML space. Elements and attributes that belong to the BXML space cannot be included within this tag, except for the `b:behavior` attribute extending HTML elements, and also the `s:innercontent` tag. For more information, see *Behaviors [p. 109]* and *Backbase Walkers [p. 39]* .

7.2.1 s:htmlstructure

The `s:htmlstructure` tag encompasses XHTML tags and their attributes, and the `<s:innercontent />` which acts as a placeholder for the contents of the custom tag that invokes this structure.

©2004-2006 Backbase B.V., All Rights Reserved.

The `<s:innercontent />` must always have an XHTML element as its parent.

The `s:htmlstructure` tag must have a `b:name` attribute. The value of the `b:name` attribute becomes the custom tag name that is used to invoke the user-defined control. The following example shows the HTML structure for a button:

Example 7.3 - User-Defined Controls

```
<s:htmlstructure b:name="c:button">
    <div style="position:relative;">
        <div class="c-button-l"></div>
        <div class="c-button-m">
            <s:innercontent />
        </div>
        <div class="c-button-r"></div>
    </div>
</s:htmlstructure>
```

To use the control, using the following syntax:

Example 7.4 - Using a Control

```
<c:button>The content of the button...</c:button>
<p>At runtime, the content of the
    <code>&lt;c:button /&gt;</code>
tag is placed in the
    <code>&lt;s:innercontent /&gt;</code>
tag     of the user-defined control.
</p>
```

The BXML namespace is reserved and may not be used for custom tags. When you create controls, create them in a namespace other than `b:` or `s:`. For example, `s:htmlstructure b:name="b:deck"` is not allowed, but you can introduce a namespace of your own, for example `s:htmlstructure b:name="c:deck"`.

7.2.2 Applying Default Behaviors

The `s:default` tag is used to give tags a default representation. For example, the Backbase control use the `s:default` tag for setting default styles and behaviors for certain tags.

When using a skin, tooltips, modals, infoboxes and other widgets do not need special classes or behaviors, since the skin already takes care of the look & feel and behavior of your web application's components.

The `s:default` tag requires the following attributes:

- `b:tag` - defines the tag name (including its namespace prefix) for which the `s:default` tag's attributes should apply to.
- `b:attribute` - a string with the name of an element's attribute (accepts the values `b:behavior` or `class`).
- `b:value` - contains a string with the value to set.

The following example illustrates an infobox widget, which follows a set behavior:

Example 7.5 - Setting Defaults

```
<s:default b:tag="b:infobox" b:attribute="b:behavior" b:value="b-infobox" />
```

In the example above, your infobox widget would follow a specific behavior named 'b-infobox'.

Another example to illustrate a different application, a modal box widget following a set style:

Example 7.6 -

```
<s:default b:tag="b:modal" b:attribute="class" b:value="b-modal" />
```

©2004-2006 Backbase B.V., All Rights Reserved.

In the example above, your modal box would be rendered according to the specific CSS 'b-modal' class properties.

Some widgets require a default skin, which contains the basic style and behavior definitions. These widgets are the slider, the datepicker, the modal dialog, the infobox and the tooltip.

7.3 User-Defined Attributes

BXML is an XML-based language which means that you can add your own attributes to elements, defined within a unique namespace. In the following example, the `b:tree` control has custom attributes defined in the `c:` namespace. You must then remember to include a namespace declaration in your document, for example `xmlns:c="http://www.mysite.com/c"`. As with any other attribute, you can target these elements using XPath and read out their values. In the example, clicking the tree node triggers an event handler defined within a behavior in which the following tasks are defined:

- The first task is a `load` command; the `url` attribute uses curly brackets {} to indicate to the BPC to resolve the XPath expression and return the result as a String. In this case, it targets the custom attribute `c:myurl` and returns the value `data.xml`. The `b:destination` indicates where the contents of the file are loaded; it resolves the value of the custom attribute `c:mytarget`, which itself contains an XPath expression targeting the element whose `id` is `welcome`, and selects the `h2` child element of this element.
- The second task is an `s:render` that adds a textnode to the `h2` element, the value of which is also derived by evaluating an XPath expression (that targets the value of the `b:label` attribute, in the case `Films`.

Example 7.7 - User-Defined Attributes

```
<b:tree b:label="Films" b:behavior="loadindata" c:mytarget="id('welcome')" c:myurl="data.xml"
/>
<s:behavior b:name="loadindata">
    <s:event b:on="command">
        <s:task b:action="load" b:url="{@c:myurl}" b:destination="xpath(@c:mytarget)/div" />
        <s:render b:destination="xpath(@c:mytarget)/h2">
            <s:textnode b:label="{@b:label}" />
        </s:render>
    </s:event>
</s:behavior>
<div id="welcome">
    <h2></h2>
    <div></div>
</div>
```

Where `data.xml` contains:

Example 7.8 - User-Defined Attributes

```
<?xml version="1.0" ?>
<table xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
<thead>
    <tr>
        <th>Movie</th>
        <th>Director</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>The Seven Samurai</td>
        <td>Akira Kurosawa</td>
    </tr>
    <tr>
        <td>Donnie Darko</td>
        <td>Richard Kelly</td>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
    </tr>
    <tr>
        <td>The Godfather</td>
        <td>Francis Ford Coppola</td>
    </tr>
</tbody>
</table>
```

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 8 - Defining Data Bindings

8.1 Templating

The BXML templating mechanism is used to transform a piece of XML from its original format, into another XML format, such as XHTML, BXML, and so on. The transformation is performed using a stylesheet, and the result moved to a specific target in the application. The tags used in BXML templating are the same tags as those used in XSLT, except the tags live in the `s` namespace. Therefore, if you are familiar with XSLT, the BXML templating mechanism is intuitive.

The BXML templating mechanism has three key components: a **Data Source**, a **Stylesheet**, and a **transform** command. The data source is the data to be tranformed, the stylesheet performs the transformation, and the transform command is the trigger that starts the tranformation process.

8.1.1 Data Source

The piece of XML that gets transformed by the templating mechanism is called the data source. The data source can be a separate XML file, or it can be part of the main BXML body.

Using a separate XML file as the data source is the most simple scenario. The XML must be well-formed, but it does not require to have the usual namespace declarations. It must also not be a BXML file. If you are using an application server, the XML data source file can be dynamically generated.

When the data source is part of the main BXML document, it requires a special tag to delimit it: the `s:xml` tag. You must specify the `b:name` attribute for the `s:xml` tag. You can then use the name to uniquely reference it. The contents of the `s:xml` tag must be well-formed, as with when you define the XML in a separate XML file. For example:

Example 8.1 - Datasource

```
<s:xml b:name="catalog">
    <products>
        <product>
            <name>Apple</name>
            <description>A Trendy Computer</description>
        </product>
        <product>
            <name>PC</name>
            <description>A Functional Computer</description>
        </product>
    </products>
</s:xml>
```

8.1.2 Stylesheet

A stylesheet is XML defining a set of instructions that performs transformations. The stylesheet can be a separate file, or it can be part of the main body of XML. In both cases, the `s:stylesheet` tag is the root tag, and the `b:name` attribute is used to uniquely identify it.

The contents of the stylesheet are a series of templates, which use XPath to match parts of the data source. When matched, fragments and pieces of information from the data source are then incorporated with parts of the template. The result of this process is returned to the application and inserted at the specified destination. By applying instructions in the stylesheet, you can transform the data source from its original format into almost any other format.

The following XSLT tags are supported within the `s:stylesheet` tag:

©2004-2006 Backbase B.V., All Rights Reserved.

- `s:apply-templates` - has optional attribute `select` that is by default set to `node()`
- `s:attribute` - has mandatory attribute `name`, and optional attribute `select`
- `s:call-template` - has mandatory attribute `name`
- `s:choose`
- `s:copy-of` - has mandatory attribute `select`
- `s:element` - has mandatory attribute `name`
- `s:for-each` - has mandatory attribute `select`
- `s:if`
- `s:otherwise`
- `s:template`
- `s:value-of` - has optional attribute `select` that is by default set to `*`
- `s:when` - has mandatory attribute `test`

Example 8.2 - Defining Stylesheets

```
<s:stylesheet b:name="product">
    <s:template b:match="/">
        <div style="height: 500px; border-width: 1px; border-style: solid;">
            <s:apply-templates b:select="products/product" />
        </div>
    </s:template>
    <s:template b:match="product">
        <div>
            <s:attribute b:name="innerHTML" b:select="name" />
        </div>
    </s:template>
</s:stylesheet>
```

The tags used in the Backbase templating mechanism are similar to those in the XSLT standard. For more information about XSLT instructions used in the stylesheet, consult an XSLT reference guide. For example: W3Schools [http://www.w3schools.com/xsl/default.asp] .

8.1.3 Transform Command

The `transform` command is used for rendering templating stylesheets. It specifies what stylesheet to use to transform what piece of data, and where in the document the result is put. The `transform` command has the following attributes:

- `b:stylesheet` - specify the name of the stylesheet using the syntax: `$<stylesheetName>`
- `b:datasource` - specify the name of the data source using the syntax: `$<dataSourceName>`
- `b:destination` - an XPath statement used to indicate the target element in the document into which the result of the transformation is placed. For more information, see *XPath Axis [p. 131]*
- `b:mode` - specifies how the result of the transformation is placed relative to the target. The attribute can have the values: `asfirstchild`, `aslastchild`, `before`, `replace`, `replacechildren`. The default value is `replace`.

Example 8.3 - Transforming Data Using Stylsheets

```
<div id="output">
    <s:event b:on="construct">
        <s:task b:action="transform" b:stylesheet="$stylesheet" b:datasource="$datasource"
b:destination="id('output')" b:mode="aslastchild" />
    </s:event>
</div>
```

8.1.4 Loading Data and Stylesheets from the Server

You can load XML data and XSLT stylesheets from the server and access them from variables for client-side transformations. You can use this functionality to do transformations client-side, enhancing the responsiveness of a Backbase front-end. Since the Backbase

©2004-2006 Backbase B.V., All Rights Reserved.

templating system is based on XSLT, the syntax on the client is the same as that used for heavier transformations on the server-side.

The following code example is taken from the Backbase petstore starter kit. It shows how XML data and XSLT stylesheets are loaded using a `load` command. Following the load, you can call the stylesheet and datasource using variables (`$name`).

Example 8.4 - Loading Data and Stylesheets

```
<s:task b:action="load" b:url="stylesheet.xml" b:destination="." />
<s:task b:action="load" b:url="catalog.xml" b:destination="." />
<s:task b:action="transform" b:stylesheet="$products" b:datasource="$catalog"
b:destination="id('main-content')" b:mode="aslastchild" />
```

Where the `stylesheet.xml` contains the following:

Example 8.5 - Contents of stylesheet.xml

```
<s:stylesheet b:name="products">
    <s:template b:match="/">
        <b:deck style="height: 400px;">
            <s:apply-templates b:select="categories/category" />
        </b:deck>
    </s:template>
    <s:template b:match="category" />
    <!--see the Backbase\3_1_4\starterkits\petshop\stylesheet.xml file for the complete
stylesheet-->
</s:stylesheet>
```

And where `catalog.xml` contains the following:

Example 8.6 - Contents of catalog.xml

```
<s:xml b:name="catalog">
    <categories>
        <category>
            <id>DOGS</id>
            <name>Dogs</name>
            <products>
                <product>
                    <id>dog1-bulldog</id>
                    <name>Bulldog</name>
                    <desc>                          General Appearance: The Bulldog is of
medium size and has a smooth coat, a heavy, thick-set, low-swung body, a massive short-faced
head, wide shoulders and sturdy limbs.                          Height: 31-40 cm.
                         Weight: 24-25kg.                          Temperament: The bulldog
is not a vicious dog breed and gets along well with both humans and sometimes other dog
breeds. Bulldogs are very friendly, but stubborn and protective.</desc>
                    <items>
                        <item>
                            <id>EST-6</id>
                            <desc>Medium size non-sporting dog</desc>
                            <listCost>18.5</listCost>
                            <unitCost>12.0</unitCost>
                            <image>dog1.jpg</image>
                            <attribute>Male Adult</attribute>
                        </item>
                        <item>
                            <id>EST-7</id>
                            <desc>Medium size non-sporting dog</desc>
                            <listCost>18.5</listCost>
                            <unitCost>12.0</unitCost>
                            <image>dog1.jpg</image>
                            <attribute>Female Puppy</attribute>
                        </item>
                    </items>
                </product>
                <!--see the Backbase\3_1_4\starterkits\petshop\catalog.xml file for the
complete datasource-->
            </products>
        </category>
    </categories>
</s:xml>
```

©2004-2006 Backbase B.V., All Rights Reserved.

8.2 Browser Templating

Browser templating involves using the XML Parser built into the client browser to transform a piece of XML from its original format, into another XML format, such as XHTML, using XSLT. To implement browser templating in Backbase, you need to define the following:

- A datasource that specifies the data to be tranformed
- A stylesheet that performs the transformation on the datasource
- And use the `xsl-transform` command that triggers the tranformation process

When you implement browser templating, you must take into account the templating capabilities of the client browser:

- Internet Explorer 6 and Mozilla has full XML support, including Namespaces, Style sheets in CSS, and XSLT 1.0. The built-in XML Parser is based on both the W3C XSLT 1.0 and the W3C XPath 1.0 Recommendations.
- Internet Explorer 5.5 has XML support; depending on the XML version installed, some support XSLT 1.0 and some do not.
- Internet Explorer 5 also has XML support, but the XSL part is NOT compatible with the official W3C XSL Recommendation.

Backbase provides the `$bpc_xsltParser` XPath variable to test for browser template support (whether the XML version supports XSLT 1.0 or XSL). You need to define alternative XSLTs to use, depending on the result of the test, to perform the transformation.

Implementations of XSLT 1.0 can differ slightly between Mozilla and Internet Explorer.

8.2.1 Implementing Browser Templating

This section describes how you can implement browser templating capabilities in your Backbase application, based on a simple example. Start by creating a start-up file with the following content:

Example 8.7 - index.html

```
<xmp b:backbase="true" style="display:none">
    <s:execute>
        <s:task b:action="show" />
    </s:execute>
    <div b:behavior="browsertemplating"></div>
    <s:behavior b:name="browsertemplating" b:focusgroup="true">
        <s:event b:on="construct">
            <s:variable b:name="data" b:scope="local" />
            <s:variable b:name="xslt" b:scope="local" />
            <!--s:variable b:name="string" b:scope="global"/-->
            <s:choose>
                <s:when b:test="$bpc_xsltParser">
                    <s:task b:action="load" b:destination="$xslt" b:url="table.xslt" />
                </s:when>
                <s:otherwise>
                    <s:task b:action="load" b:destination="$xslt" b:url="table-ie5.xslt" />
                </s:otherwise>
            </s:choose>
            <s:task b:action="load" b:destination="$data" b:url="films.xml" />
            <s:task b:action="string2xml" b:variable="$xslt" />
            <s:task b:action="string2xml" b:variable="$data" />
            <s:task b:action="xsl-transform" b:stylesheet="$xslt" b:datasource="$data"
b:destination="." />
            <!--s:task b:action="set" b:target="@innerHTML" b:value="{$string}"/-->
        </s:event>
    </s:behavior>
</xmp>
```

The `index.html` start-up file defines a `construct` event handler within a behavior. The

©2004-2006 Backbase B.V., All Rights Reserved.

event handler is triggered when the application is constructed,has the following main sections:

- **Global variables declaration** - called `data`, `xslt`, and `string`
- **Browser test** - the `$bpc_xsltParser` XPath variable used to test what XML parser the browser supports and depending on the result loads the respective XSLT and puts the resulting string into the `$xslt` variable
- **Task commands** - the last defines commands to:
  - Load the datasource `films.xml` and put the resulting String into the `$data` variable
  - Convert the `$xslt` variable from a String to XML
  - Convert the `$data` variable from a String to XML
  - Transform the contents of the `$data` variable using the `$xslt` variable and moved to a specific destination target in the application (in this example, the context node)

Because the result of a `load` command is a String, you have to convert the String to XML to make an XML document that the browser can use for templating.

For better performance, it can be useful to set the String in the HTML space, instead of the BXML space. To do this, in the above example uncomment the `s:variable b:name="string" b:scope="global"/` declaration and the `s:task b:action="set" b:target="@innerHTML" b:value="{$string}"/`. The `xsl-transform` then needs to put the result of the transformation into the global variable, `b:destination="$string"`, which is used by the `set` action to push the result into the HTML space.

The following is the `table.xslt` stylesheet used in the example above for an XSLT 1.0 compliant browser to transform the data:

Example 8.8 - table.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" b:name="stylesheet"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:b="http://www.backbase.com/b">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" />
<xsl:template match="/">
    <table style="width: 100%;" cellspacing="0" cellpadding="0">
        <tbody>
            <xsl:apply-templates select="*/*">
                <xsl:sort select="spinner" order="ascending" data-type="number" />
            </xsl:apply-templates>
        </tbody>
    </table>
</xsl:template>
<xsl:template match="*">
    <tr>
        <xsl:for-each select="*">
            <xsl:value-of select="." />
        </xsl:for-each>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

The following is the `table-ie5.xslt` stylesheet used in the example above for browser that is not fully XSLT 1.0 compliant to transform the data:

Example 8.9 - table-ie5.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns:b="http://www.backbase.com/b">
<xsl:template match="/">
    <table style="width: 100%;" cellspacing="0" cellpadding="0">
        <tbody>
            <xsl:apply-templates select="*/*" order-by="+ number(spinner)" />
        </tbody>
    </table>
</xsl:template>
<xsl:template match="*">
    <tr>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <xsl:for-each select="*">
            <xsl:value-of select="." />
        </xsl:for-each>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

Note the difference between the XSLTs: namely, the different namespace declarations, and the `xsl:output` and the `xsl:sort` tags available in XSLT 1.0.

The following is the `films.xml` datasource used in the example above:

Example 8.10 - films.xml

```
<films>
    <film>
        <title>The Seven Samurai</title>
        <director>Akira Kurosawa</director>
        <genre>Drama</genre>
        <language>Japanese</language>
    </film>
    <film>
        <title>City of God</title>
        <director>Fernando Meirelles</director>
        <genre>Drama</genre>
        <language>Portuguese</language>
    </film>
    <film>
        <title>The Godfather</title>
        <director>Francis Ford Coppola</director>
        <genre>Crime</genre>
        <language>English</language>
    </film>
    <film>
        <title>Carlito's Way</title>
        <director>Brian De Palma</director>
        <genre>Crime</genre>
        <language>English</language>
    </film>
    <film>
        <title>Apocalypse Now</title>
        <director>Francis Ford Coppola</director>
        <genre>War</genre>
        <language>English</language>
    </film>
</films>
```

Firefox has a known problem running files using the `.xslt` extension on Apache web server, when the `Content-Type` is `application/xslt+xml`. A solution for this, when using the `.xslt` extension on the Apache Web server, is to edit the content-type `application/xml` in the `mime.types` file in the `conf` directory of Apache to add the xslt extension: `xml xsl xslt`.

Alternatively, you can use the `.xsl` extension which has the normal `application/xml` content-type.

8.3 Table Sorting

In BXML, you can extend normal tables with *sorting* functionality. If you want to make use of table sorting with BXML, the rows in tables must always be nested in the XHTML `thead` or `tbody` elements, never as direct children of the `table` element.

The listview control also presents a sortable table that you can scroll through. For more information, see *Listview [p. 79]* .

8.3.1 Sorting on Table Cell Values

©2004-2006 Backbase B.V., All Rights Reserved.

Sorting in a table means organizing all the rows based on the value of the `td` elements contained therein. Usually sorting takes place when the user clicks a column heading, issuing in the background a `sort` command on a `td` element that is part of a `tr` within a `thead`. You do not need to specify a target when the command is defined for such a `td`. When calling the `sort` command from anywhere else in the interface, the `b:target` attribute should point to the `id` of a valid column heading.

Example 8.11 - Sorting on Cell Values

```
<table class="TABLE-table" sortable="true">
    <thead>
        <tr>
            <td b:action="sort" b:target="." id="name_header">Name</td>
            <td b:action="sort">Email</td>
            <td b:action="sort">Age</td>
        </tr>
    </thead>
    <tbody>
        <tr b:action="select-deselect" b:target=".">
            <td>Michelle Werner</td>
            <td>michelle.werner@atwr.com</td>
            <td b:sortvalue="32">thirty-two</td>
        </tr>
        <tr b:action="select-deselect" b:target=".">
            <td>Staci Hamamoto</td>
            <td>staci@stardust.universe.net</td>
            <td b:sortvalue="26">twenty-six</td>
        </tr>
    </tbody>
</table>
<p />
<button b:action="sort" b:target="id('name_header')">Sort     by name</button>
```

8.3.2 Sorting on an Alternative Value

Normally sorting takes place based on the text value of the contents of the `td` elements. Alternative sorting values can be specified using the `b:sortvalue` attribute. This can be useful in cases where a unit, for instance currency, is prefixed to the textual value of a table cell.

Example 8.12 - Sorting on Alternative Values

```
<table>
    <thead>
        <tr>
            <th b:action="sort">Item</th>
            <th b:action="sort">Price</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Stamps (25pcs)</td>
            <td b:sortvalue="15.00">€ 15.-</td>
        </tr>
        <tr>
            <td>Pen</td>
            <td b:sortvalue="1.75">€ 1.75</td>
        </tr>
        <tr>
            <td>CD-Recordables (50pcs)</td>
            <td b:sortvalue="20.00">€ 20.-</td>
        </tr>
    </tbody>
</table>
```

8.3.3 Sorting on Numerical Values

If either the text value of the cell or the value of the `b:sortvalue` attribute can be successfully converted to a floating point or integer number, the sorting routine will auto-

©2004-2006 Backbase B.V., All Rights Reserved.

matically take the numerical value to issue the comparison.

The following code would allow the BPC to use the `b:sortvalue` to sort the "Age" column, instead of the textual value of the table cell:

Example 8.13 - Sorting on Numerical Values

```
<td b:sortvalue="32">thirty-two</td>
```

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 9 - Client-Side Data and Business Logic

9.1 Defining Task Execution

You can declaratively implement client-side business logic using the `s:task` tag:

- Conditional logic using the `s:choose`, `s:when`, and `s:otherwise` tags
- Conditional tests using the `b:test` tag
- Sequences of tasks using `s:tasklist`, `s:parallel`, and `s:sequential`

9.1.1 Defining Tasklists to Group Commands

You can wrap an event handler in an `s:tasklist` that defines a series of commands (`s:task` tags) that are carried out when the event is triggered. In other words, the `s:tasklist` tag is a container for a sequence of commands. The `s:tasklist` contains `s:task` tags that are executed when the tasklist is called. You can call the tasklist from an event, or by using the `execute` command:

- You call a tasklist from an event handler using the syntax `b:on<EventName>="<myTaskListName>"`,
- You call the tasklist using the `execute` command; use the `b:tasklist` attribute to specify the name of the tasklist to execute:

  ```
  <code>b:action="execute" b:tasklist="taskListName"</code>
  ```

  The name of a tasklist should be unique. If multiple tasklists with the same name exist, the last defined tasklist is executed.

For example:

Example 9.2 - Defining Tasklists

```
<input type="text" b:oninvalid="dob_invalid" />
<s:tasklist b:name="dob_invalid">
    <s:task b:action="show" b:target="id('dob_invalid_msg')" />
    <s:task b:action="hide" b:target="id('dob_required_msg')" />
</s:tasklist>
```

9.2 Client-Side Validation

The validation requirements of an application are specific for each application. Backbase provides a generic solution for standard form and field validation, and provides a custom validation file for the most frequently occuring field constraints:

- **Default Validation Behavior** - Backbase provides default validation behavior to validate forms and fields in the `default.xml` file, located in your Backbase installation `Backbase/3_1_4/controls` directory.
- **Custom Validations** - the `forms.xml` located in your Backbase installation `C:/Backbase/3_1_4/backbase/controls` directory contains a BXML field validation library you can use directly, or as a basis with which your create your own custom validation.
  You can set client-side constraints on form fields, or specify a pattern that a field must satisfy using regular expressions. The user can submit the form, based on the whether the requirements are met or not. You can also provide visual hints for correct data entry.

The `default.xml` is always available unless you set the `b:skinbase` attribute on the `body` tag to `none`.

In the following example, four input fields are defined (see the the `b:on="validate"` event):

©2004-2006 Backbase B.V., All Rights Reserved.

Example 9.3 - Validation

```
<s:include b:url="../controls/forms.xml" />
<form action="data/succes.xml" method="get" b:destination=".">
    <table>
        <tbody>
            <tr>
                <td>Name:</td>
                <td>
                    <input b:required="true" name="name" type="text" />
                </td>
            </tr>
            <tr b:behavior="form-field-text">
                <td>Year of Birth:</td>
                <td>
                    <input b:on="validate" type="text" name="age" b:validation="year" />
                </td>
            </tr>
            <tr b:behavior="form-field-text">
                <td>Email Address:</td>
                <td>
                    <input b:on="validate" type="text" name="email" b:validation="email" />
                </td>
            </tr>
            <tr>
                <td>Annual Income:</td>
                <td>
                    <select name="income">
                        <option value="">Choose Income</option>
                        <option value="10_19">$10k to $19k</option>
                        <option value="20_29">$20k to $29k</option>
                        <option value="30_39">$30k to $39k</option>
                        <option value="40_49">$40k to $49k</option>
                        <option value="50_plus">$50k+</option>
                    </select>
                </td>
            </tr>
            <tr>
                <td />
                <td>
                    <input type="submit" value="submit" />
                </td>
            </tr>
        </tbody>
    </table>
</form>
```

Example 9.4 - Response File

```
<xml version="1.0" encoding="UTF-8" />
<div>      The form has been submitted.</div>
```

If you want to use the functionality in the `forms.xml`, you must use `s:include` to load the file. Default behaviors are then defined for the elements `form`, `input`, `select`, and `textarea`.

Backbase provides the following attributes for validating form fields:

- `b:required` - specify whether the field is mandatory (`true`) or optional `false`
- `b:valid` - sets the state of a field to: `true`, `false`, or `required`
- `b:pattern` - specify a regular expression (standard JavaScript library)

Backbase provides the following events for form fields:

- `valid` - sets the `b:valid` attribute to true
- `invalid` - sets the `b:valid` attribute to false
- `required` - sets the `b:valid` attribute to required
- `validate` - checks fields with `b:required` attribute, executes `b:pattern` attribute, and puts the result back in the `b:valid` attribute

Backbase provides the following events for forms (the implementation is in the `default.xml` file):

- `submit` - if all the form fields are valid, then the form is submitted. If invalid, an

©2004-2006 Backbase B.V., All Rights Reserved.

alert is shown

The super tag ensures the standard event handler is also triggered:

Example 9.5 - Validate Event

```
<input type="text" b:required="true" />
<s:event b:on="validate">
    <s:choose>
        <s:when b:test="@value &gt;10">
            <s:setatt b:valid="true" />
            <s:task b:action="trigger" b:event="valid" />
        </s:when>
        <s:otherwise />
    </s:choose>
    <s:super />
</s:event>
```

9.3 Defining Variables

A variable is an item of data named by an identifier and has a scope that determines where the variable can be used. This section describes how you can create, access, and assign variables in BXML and from JavaScript. The following variables are available in Backbase:

- *BPC XPath variables* - Backbase provides a large number of predefined global and local variables. These variables are documented in the **BXML Reference** PDF.
- *User-defined variables* - you can declare your own variables using the `s:variable` element defined within an event handler.
- *Backbase JavaScript variables* - allow you to get the current selected element and to access user-defined global, local, and tag variables from inline JavaScript. For more information, see *Using the s:script element [p. 129]*

You can access BPC XPath variables and user-defined variables using the `$` sign and the name of the variable. You can use the `assign` command to assign a new value (or node-set) to a variable. You can test whether a Backbase variable has been declared using the Backbase custom XPath function `declared()` (returns `true` if the variable has been declared, otherwise `false`).

The `s:variable` tag has the following attributes:

- `b:name` - specify a logical name for the variable
- `b:select` - specify the element that is selected (by reference) using an XPath statement
- `b:scope` - specify where the variable can be used:
  - `local` - (the default) a local variable is available within the scope of the element in which is triggered, and is passed automatically along with any event calls within the event handler. The variable is cleared from memory when the original execution thread ends.
  - `global` - a global variable is available throughout your application and exists for the lifespan of the application.
  - `tag` - a tag variable is available within the scope of the element in which it is defined, and exists for the lifespan of the application. You can access the variable by targeting the element in which it is defined.

A variable is a reference (a pointer) to a node-set; assigning a new value to the variable therefore updates the node-set referred to in the variable. For example, in the following code a global variable is set when the page is constructed that holds a reference to the sibling `div` element (and its children). Clicking the link triggers a `set` command that targets the variable, specifically a text value of a child node. It results in the node-set being updated with the value.

Example 9.6 - Variables Are By Reference

```
<div>
    <s:event b:on="construct">
        <s:variable b:name="byreference" b:select="following-sibling::div[1]"
b:scope="global" />
    </s:event>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
</div>
<div>Backbase
    <div>Rich
        <div>Internet
            <div>Applications</div>
        </div>
    </div>
</div>
<a b:action="set" b:target="$byreference/div/div/text()" b:value="RIA">Click here to see the
node-set updated</a>
```

Global variable names must be unique within the context of the entire application, and local and tag variable names must be unique within the context of the element in which they are defined. Conflicts between variable names, defined using different scopes, are resolved in the order of importance: `tag`, `local`, `global`.

9.3.1 Global Variables

Global variables are available within the scope of your entire application. In the following example, you declare a global variable within the link element `command` event handler:

Example 9.7 - Declaring a Global Variable

```
<a>
    <s:event b:on="command">
        <s:variable b:name="test_global" b:select="'Hello World!'" b:scope="global" />
    </s:event>
     Click this link to create a global variable
</a>
```

When you have declared the variable, you can access from anywhere in your application as follows:

Example 9.8 - Getting a Global Variable

```
<a b:action="alert" b:value="{$test_global}">Click here to see the value contained in the
variable</a>
```

You can assign a new value to a global variable using the `assign` command as follows:

Example 9.9 - Assigning a New Value to a Variable

```
<a b:action="assign" b:target="$test_global" b:select="'Hello Mum!'">Click here to assign a
new value to the variable</a>
```

9.3.2 Local Variables

A local variable is available within the scope of the element in which is triggered, and is passed automatically along with any event calls within the event handler. In this way, local variables allows you to pass parameters from one behavior or event handler to another. As a local variable is local to the current processing thread, when you trigger another event in your application within the context in which the variable was created, the local variable will also be available in the called event. (A local variable is cleared from memory when the original execution thread ends.)

The following example contains two link elements; the first link element has a `mouseenter` event handler that declares a local variable and triggers a `command` action in the second link; the local variable is passed on to the second link and is available in its event handler which puts the local variable value in a alert box. Clicking the second link, however, results in an XPath error because the local variable is not available within the context of that element.

Example 9.10 - Declaring Local Variables

```
<a>
    <s:event b:on="mouseenter">
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <s:variable b:name="test_local" b:select="'Hello World!'" b:scope="local" />
        <s:task b:action="trigger" b:event="command" b:target="id('trg1')" />
    </s:event>
     Click to register and trigger alert
</a>
|
<a id="trg1">
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{$test_local}" />
    </s:event>
     Click here to try and access the local variable
</a>
```

You can assign a new value to a local variable using the `assign` command as follows:

Example 9.11 - Assigning a New Value to a Variable

```
<a>Click here to assign a new value to the variable
    <s:event b:on="command">
        <s:task b:action="assign" b:select="'Hello Mum !'" b:target="$test_local" />
        <s:task b:action="trigger" b:event="command" b:target="id('trg1')" />
    </s:event>
</a>
```

9.3.3 Tag Variables

A tag variable is available within the scope of the element in which it is defined, and exists for the lifespan of the application.

In the following example, a tag variable is set when the page is constructed that holds a reference to the sibling `div` element (and its children). Clicking the link triggers an `alert` command that targets the tag variable and puts the value in an alert box:

Example 9.12 - Declaring Tag Variables

```
<div id="tagvar">
    <s:event b:on="construct">
        <s:variable b:name="test_tag" b:select="following-sibling::div[1]" b:scope="tag" />
    </s:event>
</div>
<div>Backbase
    <div> Rich
        <div> Internet
            <div>Applications</div>
        </div>
    </div>
</div>
<a>Click here to show me the tag variable (must target the element on which the tag is
defined)
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{id('tagvar')/$test_tag}" />
    </s:event>
</a>
```

You can assign a new value to a tag variable using the `assign` command as follows (note you must target the element in which the variable is defined):

Example 9.13 - Assigning a New Value to a Variable

```
<div id="tagvar2">Powered by BPC</div>
<a>Click here to show me the tag variable (must target the element on which the tag is
defined)
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{id('tagvar')/$test_tag}" />
        <s:task b:action="assign" b:select="id('tagvar2')" b:target="id('tagvar')/$test_tag"
/>
        <s:task b:action="alert" b:value="{id('tagvar')/$test_tag}" />
    </s:event>
</a>
```

The following example demonstrates that you can use the same tag variable name within

©2004-2006 Backbase B.V., All Rights Reserved.

different elements:

Example 9.14 - Using Same Tag Names

```
<s:behavior b:name="x">
    <s:event b:on="construct">
        <s:variable b:name="tag" b:select="." b:scope="tag" />
    </s:event>
</s:behavior>
<div id="div1" b:behavior="x">Backbase
    <div id="div2" b:behavior="x"> Rich
        <div id="div3" b:behavior="x"> Internet
            <div id="div4" b:behavior="x">Applications</div>
        </div>
    </div>
</div>
<a b:action="alert" b:value="{id('div1')/$tag}">Get 1</a>
<a b:action="alert" b:value="{id('div2')/$tag}">Get 2</a>
<a b:action="alert" b:value="{id('div3')/$tag}">Get 3</a>
<a b:action="alert" b:value="{id('div4')/$tag}">Get 4</a>
```

9.3.4 Using Variables for Transformations

Variables are also particularly useful for performing transformations. BXML is an XML based language, therefore you can load XML into variables and then manipulate the content by performing actions such as `remove` and `set`. In the following example, you load the XML from an external file into a variable, convert the contents to XML, remove all `<genre>` elements, and set the text of one of the elements. At each stage, an alert box shows you the result.

Example 9.15 - Performing Transformations

```
<div>Click here to see transformations</div>
<s:event b:on="command">
    <s:variable b:name="data" />
    <s:task b:action="load" b:url="data/films.xml" b:destination="$data" />
    <s:task b:action="alert" b:value="{$data}" />
    <s:task b:action="string2xml" b:variable="$data" />
    <s:task b:action="remove" b:target="$data//genre" />
    <s:task b:action="alert" b:value="{$data}" />
    <s:task b:action="set" b:target="$data/films/film[1]/title/text()" b:value="The 5th
Element" />
    <s:task b:action="alert" b:value="{$data}" />
</s:event>
```

Where the content of the `films.xml` file is follows:

Example 9.16 - films.xml

```
<?xml version="1.0" ?>
<films>
    <film id="1">
        <title>The Fifth Element</title>
        <release>1997</release>
        <description>250 years in the future, life is threatened by the arrival of Evil, and
only "The Fifth Element" can save the world.</description>
        <genre>Action</genre>
        <genre>Adventure</genre>
        <genre>Sci-Fi</genre>
        <genre>Thriller</genre>
    </film>
    <film id="2">
        <title>Blade Runner</title>
        <release>1982</release>
        <description>In a future where genetically manufactured beings called replicants are
used for dangerous and degrading work in Earth's off-world colonies, specialist police units
–             Blade Runners – hunt down and retire (kill) escaped replicants on
Earth.</description>
        <genre>Sci-Fi</genre>
        <genre>Drama</genre>
        <genre>Action</genre>
```

©2004-2006 Backbase B.V., All Rights Reserved.

```
        <genre>Thriller</genre>
    </film>
    <film id="3">
        <title>American Beauty</title>
        <release>1999</release>
        <description>A 1999 drama film that explores themes of love, freedom, self
liberation, family, and the American Dream.</description>
        <genre>Drama</genre>
    </film>
</films>
```

For information on using variables in XSLT transformations, see *Templating [p. 114]* and *Browser Templating [p. 117]* .

9.3.5 Getting and Setting Variables from JavaScript

You can access user-defined global, local, and tag variables from inline JavaScript (the `s:script` element) using the Backbase `_vars['varName'][0]` variable. You can also set a new value to a variable using the `bpc.setVariable(sName, [sValue], sScope)` function.

In the following example, a global variable is created when the page is constructed. Clicking the link executes a JavaScript alert box that contains the string value contained in the variable.

Example 9.17 - Getting Variables in JavaScript

```
<div id="my-div" b:myatt="Hello world!">
    <s:event b:on="construct">
        <s:variable b:name="myTag" b:scope="global" b:select="string(@b:myatt)" />
    </s:event>
</div>
<a>Click here to get $myTag variable from JavaScript
    <s:event b:on="command">
        <s:script>
            <!-- alert(_vars['myTag'][0]);-->
</s:script>
    </s:event>
</a>
```

Unlike when you want to access a variable in XPath, you DO NOT need to put a `$` dollar sign before the variable name.

In the following example, a global variable is created when the page is constructed. Clicking the link executes the `bpc.setVariable()` JavaScript function that assigns a new value to the variable.

Example 9.18 - Setting Variables in JavaScript

```
<div id="my-div" b:myatt="Hello world!">
    <s:event b:on="construct">
        <s:variable b:name="myTag" b:scope="global" b:select="string(@b:myatt)" />
    </s:event>
</div>
<div id="my-div2" b:action="alert" b:value="{$myTag}">Click to view $myTag variable</div>
<a>Click here to modify $myTag variable from JavaScript
    <s:event b:on="command">
        <s:script>
            <!-- bpc.setVariable('myTag', [document.getElementById('my-div2')], 'global');-->
</s:script>
    </s:event>
</a>
```

9.4 Using JavaScript

©2004-2006 Backbase B.V., All Rights Reserved.

You can add your own scripts using the following BXML elements:

- `js` command
- `s:script` tag

Backbase also provides API functions that allow you to push new XML into the BPC space with JavaScript. You can call JavaScript code from XHTML elements using BPC commands and you can also interact with the BPC runtime environment from within JavaScript. For information on the Backbase JavaScript functions, see the BXML Reference PDF.

9.4.1 Using the js command

You can add your own scripts, for example to do complex calculations or checks, and call JavaScript functions from any part of the interface using the `js` command. The content of the `b:value` attribute is evaluated when executing the `js` command. For example:

Example 9.19 - js command

```
<span b:action="js" b:value="alert('Hello world!')">Click here</span>
```

You can also use standard events to call JavaScript functions, for example:

Example 9.20 - Standard JavaScript

```
<span onclick="alert('Hello world!');">Click here</span>
```

JavaScript code running in the BXML space is run asynchronously. If you want to synchronize actions executed from BXML with actions executed from JavaScript you can use the locking mechanism (`s:lock`). If you lock the application on the root node in BXML (`<s:lock b:target="/*">`), any actions defined within the lock and wait until the JavaScript actions complete.

9.4.2 Using the s:script element

The `s:script` tag allows you to place inline JavaScript within BXML. The functions you define within the script are avaiilable within the context of the element in which the script is defined. The advantage of inline JavaScript is that you have all that local, global, and tag variables at your disposal as you have in XPath. You can access these variables using the Backbase `_vars` JavaScript variable using the following syntax: `_vars['myvar'][0]`.

All variables are nodelists and are represented as an array of values. Therefore, even if the nodelist only contains one value, you have to retrieve it from the array, for example using `[0]`. You can also access the current node using the `_current` variable (in the example below, this is the `div` element). For example:

Example 9.21 - Using JavaScript Variables

```
<div>
    <s:event b:on="construct">
        <s:variable b:name="myvar" b:select=".." b:scope="local" />
        <s:script>
            <!-- _vars['myvar'][0].style.backgroundColor = 'blue'; _current.style.color =
'white';-->
</s:script>
    </s:event>
     This text will be white on a blue background
</div>
```

The Backbase JavaScript variables (`_vars` and `_current`) can only be used within JavaScript; you cannot use them within XPath.

9.4.3 Escaping JavaScript

©2004-2006 Backbase B.V., All Rights Reserved.

You must always properly escape the inline JavaScript contained in the `s:script` tag otherwise the characters within your code causes XML parse errors. It is recommended to encapasulate the script within `s:script` tags using XML comments:

Example 9.22 - Escaping JavaScript Using XML Comments

```
<s:script><!--     Add your own JavaScript     -->
</s:script>
```

If your script contains XML sensitive characters such as `--` , you must escape using a different syntax. For example, using the syntax `alert(i--)` leads to parsing errors; instead, use `alert(i=i-1)` or `alert(i-=1)`.

Example 9.23 - Escaping JavaScript Containing Sensitive Characters

```
<span>
    <s:event b:on="command">
        <s:script>
            <!-- var i=3; alert(i-=1); alert(i=i-1); -->
</s:script>
    </s:event>
    Click to see alert with variable value
</span>
```

There are 5 predefined entity references in XML:

- `<` represents `<` (less than)
- `>` represents `>` (greater than)
- `&` represents `&` (ampersand)
- `'` represents `'` (apostrophe)
- `"` represents `"` (quotation mark)

©2004-2006 Backbase B.V., All Rights Reserved.

Chapter 10 - Appendix

10.1 Performance Tuning

This section contains general recommendations to improve performance:

- To reduce latency, include one large file instead of several small files using `s:include`. For example, bundle the controls you use and remove the controls you do not use and put them in a single file that is included.
- Internet Explorer can suffer extreme performance hits during the rendering of HTML related when you set the style width and height to 100%. This may also occur with elements that have width and height specified in pixels. To ease this problem, add the rule `div{overflow:hidden;}` to your stylesheet. (All `div` elements in the DOM tree appear to be affected by this issue; forgetting one `div` may still results in a full performance hit.
- Use a small JavaScript file, `bpc.js` for optimizing resources. For more information, see *Preloading JavaScript [p.]*

10.1.1 Preloading JavaScript

The Backbase JavaScript libraries include a small JavaScript file, `bpc.js`, that you can use for optimizing resources when placing links to Backbase Startup Files in normal HTML Web Pages. The `bpc.js` file partially loads the BPC when the web browser is using less system resources. The moment the user clicks the link that loads the Backbase Startup File, most of the resources are already loaded, therefore improving the speed of page download.

This causes a significant part of the libraries to be already cached when loading the next page, improving the download performance.

Example 10.1 -

```
<xmldecl version="1.0" encoding="UTF-8" />
<doctype />
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
    <head>
        <title>Traditional HTML Page</title>
        <script type="text/javascript" src="/3_1_4/bpc/bpc.js"></script>
    </head>
    <body>
        <h1>Traditional HTML Page</h1>
        <p>For more information, please consult the Backbase DevNet on our web site.</p>
    </body>
</html>
```

10.2 XPath 2

XPath is a language for addressing parts of an XML document which operates on the abstract, logical structure of an XML document, rather than its surface syntax. It models an XML document as a tree of nodes which are of different types, such as element nodes, attribute nodes and text nodes. XPath also provides basic facilities for manipulation of strings, numbers and booleans.

The information in this section is designed to provide an overview of the XPath Axis, functions, and operators supported by Backbase, and provides some simple XPath examples. For more information on XPath, see the BXML Reference PDF, or W3Schools XPath Tutorial [http://www.w3schools.com/xpath/] .

10.2.1 XPath Axes

©2004-2006 Backbase B.V., All Rights Reserved.

An axis defines a node-set relative to the current node. The location path consists of one or more steps, each separated by a slash, and can be absolute or relative. An absolute location path starts with a slash (/) and a relative location path does not. For example, an absolute location path `/step/step/...` and a relative location path is `step/step/...`. Each step is evaluated against the nodes in the current node-set.

The syntax for a location step is `axisname::nodetest[predicate]`, where:

- an axis (defines the tree-relationship between the selected nodes and the current node)
- a node-test (identifies a node within an axis)
- zero or more predicates (to further refine the selected node-set)

Next to the standard XPath axis, Backbase has introduced the following axis and short-cuts:

- `~-` is a shortcut for the `preceding-sibling` axis
- `~+` is a shortcut for the `following-sibling` axis
- `object` is an axis that enables you to get the values of properties of selected nodes.
- `style` is axis that enables you to get CSS properties of selected nodes.

The following table describes the XPath Axis:

Axis Name / Shorthand Notation	Description	Syntax Examples
`child::` Shorthand: If an axis is not specified, the `child::` is used implictly	Selects all or specified children of the context node.	`child::*` selects all the child elements the context node `child::p[2]` selects the second `p` child of the context node
`parent::` Shorthand: `..`	Selects the parent of the context node.	`../..` select the grandparent of the context node
`ancestor::`	Selects all or specified ancestors (parent, grandparent, and so on) of the context node. The closest ancestor (its parent) comes first in the node list.	`ancestor::*` selects all ancestors `ancestor::table` selects all table ancestors `ancestor::b:window[1]` selects the first `b:window` ancestor

`ancestor-or-self::`	Selects the context node and all, or specified, of its ancestors (parent, grandparent, and so on).	`ancestor-or-self::*`
		`ancestor-or-self::table`
		`ancestor-or-self::b:window`

`attribute::` Shorthand: `@`	Selects all attributes of the context node.	`@*`selects all the attributes of the context node `@id` selects the id attribute of the context node
`self::` Shorthand: `.`	Selects the context node.	Note: If you do not specify a `b:target`, by default the context node is selected
`descendant::` Shorthand: `//`	Selects all descendants, (children, grandchildren, and so on).	`//p` selects all `p` children of the context node
`descendant-or-self::`	Selects the context node and all its descendants,	`descendant-or-self::p` selects all `p` elements below

©2004-2006 Backbase B.V., All Rights Reserved.

Axis Name / Shorthand Notation	Description	Syntax Examples
	(children, grandchildren, and so on).	the context node and the context node itself if it is also a `p` element
`object::`	Selects the values of properties of selected nodes (Backbase custom Axis).	For example, `/div/span/object::width` selects the width value of the target node
`style::`	Selects CSS properties of selected nodes (Backbase custom Axis).	For example: `div/style::float`
`preceding::`	Selects everything in the document that is before the start tag of the current node	`preceding::`
`preceding-sibling::` Shorthand: `~-`	Selects all siblings before the context node. If the context node is an attribute node or namespace node, this axis will be empty.	`~-*[3]` returns the sibling before the context node
`following::`	Selects everything in the document after the closing tag of the current node	`following::`
`following-sibling::` Shorthand: `~+`	Selects all siblings after the context node. If the context node is an attribute node or namespace node, the axis is empty.	`~+*[1]` returns the first sibling after the context node

10.2.2 XPath Operators

An XPath expression returns either a node-set, a string, a Boolean, or a number. XPath operators behave in much the same way as similar operators in other languages. They are used to perform; arithmetic, value comparison, general comparison, node comparison, test Booleans, and there are also operators that combine sets of nodes.

The following tables is a list of supported XPath operators:

Operator	Description	Example	Return Value
|	Union that computes two node-sets	`//book | //cd`	Returns a node-set with all book and cd elements
+	Addition	`6 + 4`	10
-	Subtraction	`6 - 4`	2
*	Multiplication	`6 * 4`	24
div	Division	`8 div 4`	2
=	Equals	`price=9.80`	true if price is 9.80, false if price is 9.90
!=	Not equal	`price!=9.80`	true if price is 9.90,

©2004-2006 Backbase B.V., All Rights Reserved.

Operator	Description	Example	Return Value
			false if price is 9.80
<	Less than	`price<9.80`	true if price is 9.00, false if price is 9.80
<=	Less than or equal to	`price<=9.80`	true if price is 9.00, false if price is 9.90
>	Greater than	`price>9.80`	true if price is 9.90, false if price is 9.80
>=	Greater than or equal to	`price>=9.80`	true if price is 9.90, false if price is 9.70
or	Boolean disjunction (A or B)	`price=9.80 or price=9.70`	true if price is 9.80, false if price is 9.50
and	Boolean conjunction (A and B)	`price>9.00 and price<9.90`	true if price is 9.80, false if price is 8.50
mod	Modulus (division remainder)	`5 mod 2`	1
eq	Equals	`price eq 9.80`	true if price is 9.80, false if price is 9.90
ne	Not equal	`price ne 9.80`	true if price is 9.90, false if price is 9.80
lt	Less than	`price lt 9.80`	true if price is 9.00, false if price is 9.80
le	Less than or equal to	`price le 9.80`	true if price is 9.00, false if price is 9.90
gt	Greater than	`price gt 9.80`	true if price is 9.90, false if price is 9.80
ge	Greater than or equal to	`price ge 9.80`	true if price is 9.90, false if price is 9.70
is	Ordering/identity comparison between nodes	`../*[not(. is current())]`	all child elements of the parent element, except the current node
except	Difference of sequences considered as sets of nodes	`../* except .`	all child elements of the parent element, except the current node
intersect	Includes a node in the result if it is in both the two sets	`following::p intersect $myvar//*`	all `<p>` elements of the context node, provided they are descendents of the node in variable `$myvar>`

The following is a list of **unsupported XPath operators:**

- `castable as`
- `instance of`
- `treat as`

©2004-2006 Backbase B.V., All Rights Reserved.

- `cast as`
- `return`
- `every`
- `some`
- `then`
- `else`
- `for`
- `to`

XPath limitation: Using an XPath expression that uses an operator AND targets a hyphenated attribute or element name in which an XPath reserved word is used gives incorrect results. Examples of reserved words are operator names such as `in` or `eq`. (Using hypenation such as `@value-of` is permitted). In the following example, XPath cannot resolve the target `b:in-x` attribute:

Example 10.2 - Hypenated Attributes

```
<div>
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{boolean(@b:in-x = 'y')}" />
        <s:setatt b:in-x="y" />
        <s:task b:action="alert" b:value="{boolean(@b:in-x = 'y')}" />
    </s:event>
     @b:in-x="y"
</div>
```

10.2.3 XPath Functions

XPath includes over 100 built-in functions: there are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more. For example, the `id()` function is frequently used to select an element by its unique ID; `id('mainlayer')`.

The supported XPath functions are documented in the BXML Reference PDF. Next to the standard XPath functions, Backbase has introduced the following custom **Backbase XPath functions:**

- `current()` - this function is part of XSLT 2.0 standard, but since certain tasks are not possible without it, it was added to our implementation of XPath 2.0
- `html()` - for moving from the BXML space to the HTML space
- `bxml()` - for moving from the HTML space to the BXML space
- `xpath()` - evaluates XPath expression given as a string parameter of function
- `cookie()` - gets a cookie

The following is a list of **unsupported XPath functions:**

- `adjust-date-to-timezone`
- `adjust-dateTime-to-timezone`
- `adjust-time-to-timezone`
- `base-uri`
- `collection`
- `data`
- `default-collation`
- `doc`
- `document-uri`
- `expanded-QName`
- `idref`
- `in-scope-prefixes`
- `lang`
- `namespace-uri`
- `namespace-uri-for-prefix`
- `namespace-uri-from-QName`
- `nilled`
- `normalize-unicode`
- `resolve-QName`
- `resolve-uri`

©2004-2006 Backbase B.V., All Rights Reserved.

- substract-dates
- substract-dateTimes
- unordered

10.2.4 XPath Examples

The following table provides some examples of targeting:

XPath Syntax	Explanation
`../..`	`self`, `parent`, `parent` Start from context node (`self`) and walk up two parents.
`../../div`	`self`, `parent`, `parent`, named `child div` elements Start from context node (`self`) walk up two `parents`, get the `children`, and filter out everything but the `div` elements.
`~+*[3]`	`self`, and `third following sibling` Start from context node (`self`) and retrieve the `third sibling` after the context node.
`~-*[2]`	`self`, any `second preceding sibling` Start from context node (`self`) and retrieve the `second sibling` before the context node.
`id('mylayer')/~+*[2]`	`unique id`, `second sibling` with the same element type Start from the unique ID element, and attempt to find the second sibling of the same type as the element with the unique ID. For example, if `mylayer` is a `div`, it returns the second sibling `div` element.
`../../div[2]`	`self`, `parent`, `parent`, named `child div` elements, select `second child` Start from the context node (`self`), walk up through two parents, match all `div` elements, and select the second child element in the `div` elements.
`id('mainlayer')/div[@b:state='selected']`	`unique id`, `children` that are `div` elements where attribute `b:state` is `selected` Start from the unique ID element, and return all child elements of type `div` where the attribute `b:state` is set to `selected`.
`id('mainlayer')/div [@b:state='selected' and @b:behavior='button']`	`unique id`, `children` that are `div` elements where attribute `b:state` is `selected` **and** attribute `b:behavior` is `button` Start from an the unique ID element, and return all `div` child elements where the attribute `b:state` is set to `selected` and the attribute `b:behavior` is set to `button`.
`id('mainlayer')/div[@b:state='selected and @b:behavior != 'list']`	`unique id`, `children` that are `div` elements where attribute `b:state` is

©2004-2006 Backbase B.V., All Rights Reserved.

XPath Syntax	Explanation
	`selected` and attribute `b:behavior` is **not** `list` Start from the unique ID element, and return all `div` child elements where the attribute `b:state` is set to `selected` and the attribute `b:behavior` is **not** set to `list`.
`id('mainlayer')/div[@b:behavior='button' or @b:behavior='toolbutton']`	`unique id`, `children children` that are `div` elements where attribute `b:behavior` is `button` or `toolbutton` Starts from the unique ID element, and return all `div` child elements where the attribute `b:behavior` is set to `button` or `toolbutton`.
`id('mainlayer')/@b:behavior`	`unique id`, return the value of attribute `b:behavior` Start from the unique ID element, and return the value of the attribute `b:behavior` on the element with the `unique ID`.

Use curly brackets `{}` in the values of BXML attributes to indicate that it contains an XPath expression that requires resolving; the BPC then evaluates the contents as XPath and returns the value as a String. For example, `<a id="Hello" b:action="alert" b:value="{@id}">Click me</a>` evaluates the id attribute of the context node (the current element) and returns its value in an alert dialog box.

Backbase is similar to XSLT in using single curly brackets `{xxx}` to indicate that the attribute value contains an XPath expression that requires resolving. It differs from XSLT, however, in that the value must start with the opening curly { bracket. If you want to concatenate strings, use the `concat()` function.

10.3 Browser Compatibility

The Backbase Presentation Client (BPC) runs on all major browsers (Internet Explorer, Mozilla, Firefox, and Netscape). For optimal user experience, it is recommended to use higher versions of these browsers.

Backbase supports the following browser/operating system combinations:

Web Browser	Operating System
Internet Explorer 5	Windows 98 Windows Me Windows NT
Internet Explorer 5.5	Windows 98 Windows Me Windows NT Windows 2000
Internet Explorer 6	Windows 98 Windows Me Windows NT Windows 2000 Windows 2000 Windows XP

©2004-2006 Backbase B.V., All Rights Reserved.

Web Browser	Operating System
Mozilla 1.7	Windows 98 Windows Me Windows NT Windows 2000 Windows XP Linux Kernel 2.2+ MacOS X 10.2+
Mozilla Firefox 1.0 - 1.5	Windows 98 Windows Me Windows NT Windows 2000 Windows XP Linux Kernel 2.2+ MacOS X 10.2+
Netscape 7.2	Windows 98 Windows Me Windows NT Windows 2000 Windows XP Linux Kernel 2.2+ MacOS X 10.2+
Netscape 8.0	Windows 98 Windows Me Windows NT Windows 2000 Windows XP
Camino 0.82	MacOS X 10.2+

©2004-2006 Backbase B.V., All Rights Reserved.